TURING Coding Kids

程序员妈妈带你进入Java“魔法世界”

丰富有趣的示例帮你掌握Java“咒语”

Java少儿编程

[西] 纳迪娅·阿梅西亚内·加西亚 著

李凡妮 姚均霖 译




中国工信出版集团


人民邮电出版社
POSTS & TELECOM PRESS

[西] 纳迪娅 · 阿梅西亚内 · 加西亚

软件开发质量工程师，关注最佳实践、软件开发周期和敏捷方法，尤其侧重于测试策略。她热衷于编程和教育，有两个女儿，这是她编写本书的动力所在。她以这种方式将女儿带入计算机的世界。

李凡妮

毕业于上海外国语大学西班牙语语言文学专业，曾留学阿里坎特大学，拥有丰富的口译及笔译经验。爱好旅行、烘焙和游泳。

姚均霖

毕业于上海交通大学机械自动化专业，留学欧洲，获巴黎综合理工学院和苏黎世联邦理工学院计算机科学硕士学位。曾参与美国阿贡国家实验室（ANL）和腾讯的研究项目，并在国际顶尖会议发表论文。现为第四范式数据科学家。爱好旅行、阅读和美食。

数字版权声明

TURING Coding Kids

Java
少儿编程

[西] 纳迪娅 · 阿梅西亚内 · 加西亚　著
李凡妮　姚均霖　译



人 民 邮 电 出 版 社
北 京

图书在版编目（CIP）数据

Java少儿编程 / (西) 纳迪娅·阿梅西亚内·加西亚著；李凡妮，姚均霖译. -- 北京：人民邮电出版社，2019.3

（Coding Kids）

ISBN 978-7-115-50763-1

Ⅰ. ①J… Ⅱ. ①纳… ②李… ③姚… Ⅲ. ①JAVA语言－程序设计－少儿读物 Ⅳ. ①TP312.8-49

中国版本图书馆CIP数据核字(2019)第023261号

内 容 提 要

本书以创造性、趣味性的方法讲解 Java 基本概念，从软件安装和算法基础开始，绘制一条尽可能简单的学习道路，抛弃枯燥的理论和严苛的条件，让读者从一开始就能自主构建小脚本或小程序，例如拥有自己的电子存钱罐、和计算机玩石头剪刀布游戏等。

本书适合对 Java 编程感兴趣的中小学生自学或者在家长的帮助下学习。

◆ 著　　　　[西] 纳迪娅·阿梅西亚内·加西亚
　译　　　　李凡妮　姚均霖
　责任编辑　傅志红
　责任印制　周昇亮

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
　邮编　100164　　电子邮件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　北京　　　　　　印刷

◆ 开本：787×1092　1/16
　印张：7.25
　字数：105千字　　　　2019年3月第1版
　印数：1-3 000册　　　2019年3月北京第1次印刷

著作权合同登记号　图字：01-2018-1406号

定价：39.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京东工商广登字20170147号

译者序

一直以来，我们都想翻译一本写给孩子的书，因为这类作品语言亲切、内容鲜活，读着读着你就会扑哧笑出声来。更重要的是，它字里行间富有感染力，能神奇地帮我们找回自己那颗渐渐被遗忘的童心。机缘巧合之下，终于等到了这部作品——一本写给孩子的编程启蒙书。

我们的小读者，或许你很想知道，这本书会讲什么，自己能从书里学到什么。让我来悄悄告诉你，当你打开这本书时，会进入一个叫编程的“魔法世界”，成为一名小小“魔法师”，移动鼠标、敲击键盘、施展名为 Java 的“魔法”。在这个世界中，你将发挥自己的想象力，用魔法操控电脑，把想法变为现实，创造属于自己的一片天地。

这本书将教你如何成为一名“魔法师”。你将学会踏入魔法世界的第一条咒语，让电脑屏幕显示你希望出现的文字；你将创造独一无二的电子存钱罐，存入自己的零花钱，还可以给它加上魔法保护罩，只有知道暗号的小伙伴才能打开；你将模拟火箭升空倒计时，电脑会听从你的命令从 5 数到 1；更棒的是，你能通过魔法，让计算机和你进行“石头剪刀布”对战，看看谁更胜一筹；最后，你还将学会如何加密你的机密文件，把它变成无人能破解的神秘文字，只有你知道真实含义。这些魔法其实一点都不难，只要耐着性子，踏实学习，认真钻研，我相信你一定能够掌握它们，有所收获！

这本书对两位译者来说也意义非凡。我们一个主修西班牙语、另一个主修计算机，跨越了两个大陆、七个时区。经过长达三个月的远程协作，我们最终完成了这部作品的翻译。虽然已经认识十多年，但从未想过有一天能以这种形式合作，借这个机会探索学习彼此的专业领域。我们非常感谢翻译时的相互鼓励和支持，希望在其他领域也能延续这份并肩作战的默契。

在这里还要特别感谢我们的编辑静文。她在本书的翻译过程中一直非常尽心尽责，耐心地帮助我们联系作者、核对细节、反复沟通，旨在为读者们呈现一部更好的作品。

最后，欢迎我们的小读者进入这个叫编程的“魔法世界”，祝愿你顺利成为一名“魔法师”！

李凡妮

2018 年 5 月于上海

姚均霖

2018 年 5 月于苏黎世

引言（写给大人们）

首先，非常感谢你拨冗阅读本书。我觉得你一定是想亲自了解一下，你的儿女、学生或身边的其他孩子会从本书中学到什么。

也许你已经对编程基础知识有所了解，甚至你已经是一位行家，只是希望从本书中寻找灵感，想要以创造性、趣味性的方法讲解这些知识，那么你会发现，有一些主题只会**略微带过**，因为本书更注重基础概念，而非专注于某些细节。这样能让孩子们更熟悉表达指令的方式，而不会过分关注指令背后隐藏的千万种细节和可能。

反之，如果你之前从未有过相关经历，也丝毫不用害怕。这本书不会抛弃任何人，跟着我们循序渐进，你会发现书中的概念其实清晰又简单。如果有人向你寻求编程方面的帮助，我相信你一定有能力帮他答疑解惑。此外，针对不同章节，你可能需要在计算机上做一些准备工作。为了不让你手足无措，后面有一份指南，在里面你会找到所需的所有内容。

再或者，你是一个大大方方把这本书买给自己的成年人。如果是这样，恭喜你！我相信本书也会让你很有收获。尽管我把它定位为针对 8~18 周岁青少年的 Java 入门指南，但所有内容同样也非常适用于你的情况。

你可能有几个困惑

也许你想知道，**编程对孩子们究竟有什么用**。其实我一直在问自己，难道还有什么是编程做不到的吗？

编程是一种能帮助我们搭建新思维的绝妙技能，让我们以一种更有**逻辑**的方法解决遇到的问题。它教我们对可用数据进行**排序**，完成高效处理。我总是说，学习编程和算法能开启思维和心智的开关。**程序员的思维方式**会成为一种受用终生的工具，在许多时候都发挥着至关重要的作用。

编程同样也能考验我们的**耐心**和**抗挫折能力**。有时候，编写一段代码就像是在和自己对弈。突然之间某个地方就出了问题，我们怎么也弄不明白，但是也不能轻易半途而废。于是，我们在脑袋里反复琢磨，上学路上想，洗澡的时候想，散步的时候也想……直到有一天，解决难题的方法突然闪现于脑海——我终于明白啦！**这种跨出一大步、每次遇到困难都有所成长的感觉，对所有人来说都是无比珍贵的礼物，对于成长中的孩子们就更不用说了。**

学习编程也有社交层面的作用。赶快甩开你对编程的偏见，从脑海中抹掉那些孩子们一边喝着饮料，一边抱着计算机的画面吧。无论是在线上还是线下，编程都鼓励我们相互分享、比较意见、开展协作，也推动我们组成团队、分享项目和知识，并从别人写的代码中总结错误、学习经验。在这个过程中，还可以与其他机灵的小脑袋聚在一起，集思广益。

最后还有一点也很重要，**编程是民主的体现**。它不需要任何特殊技能，也不限制最低年龄，甚至有生理缺陷也没关系。你不需要昂贵的设备，也不需要付费软件许可。任何人都可以随时学习编程。

如果你的孩子看起来更适合学习文科，你也许会疑惑，他到底适不适合学习编程。尽管外行人觉得编程像是一种面向高智商分子和技术狂人的黑魔法，但我必须要告诉你，编程既不是文科，也不是理科——它其实是一门跨学科知识，超越了教育体系在不同学科间划分的一般界限。它是一种非常强大的工具，不仅能满足科学兴趣，也可以为艺术性和创造性的思维服务。此外，虽然有点不好意思承认，但**最好的程序有时就源于懒惰**。人们不想亲自上阵，希望那些单调乏味的任务能自动执行，或者想要节约点时间，思考、创造和加工抽象思维的时机就应运而生。对每一个学生来说，编程都是必不可少的工具。

同样，我也想告诉你**本书的学习目标**。写这本书的初衷并不是想让孩子们成为

编程专家，本书也不能取代编程领域的正规学习过程。我只希望它能激发读者的好奇心，**绘制一条尽可能简单的学习道路，抛弃大量枯燥的理论和严苛的条件，让孩子们从一开始就能自主构建小脚本或小程序。**我希望编程学习者能在过程中接受挑战、培养兴趣，并收获成就感。希望他有足够的知识把想象变为现实，不要没理解原理就机械地完成任务。我相信上进心会推着他追寻更广阔的知识，我只是起了个头。如果孩子们读毕本书，想要继续深入学习下去，我就再高兴不过了。要问编程艺术中是否有放之四海而皆准的真理，那必然是学无止境。

引言
（写给刚拿起这本书的你）



恭喜你！你现在已经差不多是一名程序员啦。谢谢你的信任，把这项教学使命交给我。

通过这本书，我想揭开一个其他教科书都想要掩盖的秘密：**编程其实非常有趣**。我非常喜欢编程，因为它是这世上为数不多能让梦想变为现实的方法。（好吧，可能只是**一部分**梦想。）的确，编程不是一套完美的系统。但据我所知，还没有其他方法能像编程这样，把想法变为指令，再通过执行这条指令，把仅仅存在于脑海中的想法化为现实。

此外，对于充满好奇心的人来说，**编程给了他们一个机会，去了解身边那些目前看起来还毫不起眼的事物，知晓它们是如何运转的。**一旦掌握了一些编程知识，你就会以另外一种视角看待这个世界了——你会开始想象，自己最喜欢的电子游戏、饮料自动贩卖机、超市的台秤是如何运转的；你还想知道，自己是不是也能写出让这些设备运转的指令；或者像我一样，你不满足于此，甚至在考虑人们是否想到了所有使用那台设备的可行方式……有一些出乎意料的操作可能导致设备崩溃，你又是否能够发现它们？

总之，我试着将这本书写得生动有趣。希望你能尽快学会编写小程序，并将学习到的每个新知识点都应用其中。写入书中的都是对你来说非常基础且必要的内容，因此建议你**不要快速浏览**，**也不要跳过某些章节**，最好按照章节顺序阅读。

啊，还有一点！有时你会看到一些**灰色方框的延伸阅读**，这是我留给你的一些小线索。如果你还好奇，想要深入了解相关主题，或是觉得我教给你的远远不够，就可以继续研究这些内容。你也可以先把这些小方框放在一边，甚至置之不理——反正你说了算。

最后再给你两条建议。

如果你喜欢编程，那就别着急，一步一步来。别对自己要求太高，没有人生下来就什么都会。最重要的是，要给自己足够多的机会去尝试。

如果你不喜欢，不用不安，放下这本书吧。即使你还年轻，但生命毕竟短暂，不要浪费时间赶一趟不想搭乘的火车。

祝你旅途愉快！

我们需要准备什么呢



我们需要一台计算机来完成书中的练习。你可以使用基于 Linux 的操作系统，比如 Ubuntu（这是个很棒的选择），也可以选择任何版本的 Windows 或 MacOS。

把计算机连上网，这对你也很重要。

你需要的程序和组件

JDK，即 **Java 开发工具包**（Java™ Development Kit）。我们要保证执行程序时，计算机能够读懂 Java 代码。我们可以从 Oracle.com 免费下载 JDK。记得下载 Java SE Development Kit 8，这是本书代码的开发环境。

还要确认你下载的 JDK 版本是否与你的操作系统（Windows、Mac 等）及处理器（X86 或 X64）相匹配。如果你不确定处理器的类型，可以在计算机的信息页面查看（如系统、我的电脑、关于本 Mac，或是类似页面）。下载 JDK 后，必须配置你的**环境变量**。虽然每个操作系统中的配置过程都不相同，但它非常简单。我建议你按照 Oracle 的帮助页面提及的步骤进行配置。

文本编辑器。在做最开始的那些练习时，我推荐你使用尽可能简单的文本编辑器，比如一些功能强大的免费编辑器，像 Notepad++、Atom 和 Vim。本书示例使用的是 Notepad++ 编辑器。

命令行控制台。在 Windows 环境下，你将使用命令提示符；而在 Ubuntu 和 Mac 环境下，你使用的是终端（terminal）。强烈建议你熟悉一下“切换目录”“查看目录包含的文件”等方法。你可以在网上进行一次快速检索，了解可能需要的基本命令。

你并不需要集成开发环境（Integrated Development Environment，IDE）或是其他特别的开发软件。**本书的练习只需要文本编辑器就可以完成**。因为我认为，学习使用终端充满着乐趣，使用文本编辑器避免了在学习编程的同时还得学习 IDE。你之后有足够的时间慢慢学习这些工具。

目录

第0章

被称作黑魔法的编程



在开始阅读本书之前，恐怕你得先了解一下计算机和它的工作方式。**编程就像是和计算机对话。如果我们不理解计算机的思考方式，就很难让它听懂我们的话。**

也许你已经了解了，**程序是一段由若干命令组成的脚本，而你的计算机会根据这段脚本执行一系列操作**。但问题是计算机并不会说我们的语言，而是说一种被我们称为“机器码”的语言。更麻烦的是，每种处理器都对应着不同的机器码。机器码由若干数字组成，分别对应不同种类的操作、数值或是其他数据。无论行为有多不起眼，都会被一一罗列出来。**由于使用机器码编程非常耗时、又很复杂，因此人们发明了其他更便于使用的编程语言**，比如我们将在本书中学习的 Java。当你运行程序时，这些编程语言会被翻译成处理器能够理解的机器码。尽管 Java 语言的一小部分代码还是会令人困惑，但它其实已经是对人类来说较为“简单”的一种语言了。对了，刚才说的“翻译”是由一个叫**编译器**或是**解释器**的程序完成的。

当编译器**编译**我们所写的代码时，它会读取指令，检查代码语法是否正确（这些指令有时候也存在语法以外的其他错误）。如果代码的语法正确，编译器就会将指令翻译成机器能够理解的语言；如果代码语法不正确，那么我们会看

到错误提示，也无法编译程序……别担心，错误每天都会发生，即使是世界上最专业的程序员也不例外。

编译型还是解释型？



Java是一门解释型语言。

虽然解释器和编译器有相似之处，但它们并不是同义词。你可以上网搜索，了解更多编译型语言和解释型语言的不同之处。

书写魔法：算法

现在我们已经了解到，人们使用了一种方便理解的语言进行编程，然后将其传送给编译器翻译，使处理器能够读懂代码。

然而，我们该怎么组织这一系列的命令呢？

电脑只会照本宣科：它不会做任何没被要求做的事，它也会尽可能执行所有发送给它的命令。电脑总是需要我们告诉它要执行什么、怎么做、什么时候做和按什么顺序做，甚至还要告诉它，当某些事不能完成时，我们又期望它做些什么。为了有序、清晰地表达所有命令，我们需要一套工作方法、一种详细阐述指令的规范。于是就有了**算法**。我很赞同维基百科对算法的词条解释：**算法是一套指令，或一系列明确定义、有序且有限的规则，它允许我们通过连续的步骤完成一个活动，而这些步骤不会使执行者产生歧义。**我们可以把算法想象成传达给某个人的一系列指令，这个人必须完成某个任务却不知道怎么开始。

如果要为一名不知道怎么给植物浇水的见习园丁编写一个算法，可以这样写：

(1) 拿一个水壶；

(2) 将它放在水龙头下面；

(3) 打开水龙头，等待水流出；

(4) 让水龙头开着，直到水壶盛满 500 毫升的水；

(5) 关上水龙头，等待水流停止；

(6) 把水壶带到花园；

(7) 找到植物的位置；



(8) 在植物上方倾斜水壶，保持住，直到水全部浇完；

(9) 把水壶放好。

正如你看到的，在我们发给见习园丁的指令中，有几个十分重要的细节：

- 完成动作的**顺序**；
- 动作**什么时候结束**；
- 完成浇水后该**做什么**。

如果不注明动作顺序，见习园丁可能会先浇水，再往水壶里装水，这样植物就会因缺水而死；如果不告诉他什么时候关上水龙头，那我们的园丁朋友可能会让房子变成游泳池；如果不告诉他最后应该收好水壶，那他可能会永远把水壶挂在植物上。我们的朋友只会按照指示做事，电脑也是如此，这就是为什么算法必须要**准确、有序，并且有明确的结束**。

以上就是你开始写代码前需要了解的所有预备理论。**太棒啦！**

练习



我为你准备了一些非常简单的练习，看看你是不是正确理解了上面的内容。我们可不想让植物因缺水而枯萎。

（练习的答案在第 5 页，可别提早偷看答案哦！）

小测试

(1) **处理器可以执行你用 Java 编写的代码吗？**

a) 当然可以

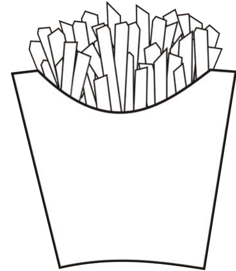
b) 根本不行

(2) 算法指的是……

a) 处理器能够理解的一段代码

b) 一系列具体、确切、有限的命令

(3) 为了理解你编写的语言，你的计算机需要（　　）的翻译



a) 处理器

b) 解释器

避免灾难的发生

下面这个算法的一些错误会给我们招来祸患，你能指出并纠正这些错误吗？

炸薯条的做法：

(1) 准备足够的土豆；

(2) 拿一把削皮刀；

(3) 开始削土豆皮；

(4) 把平底锅放在炉子上；

(5) 开火并调至中火；

(6) 将油倒入锅中；

(7) 将土豆倒入平底锅中；

(8) 炸土豆；

(9) 将土豆从平底锅盛出并放在盘子上。

编写你自己的算法

这是给勤奋学生的附加练习。根据下述任务创建指令序列，即算法：

- 熨一件衬衫；
- 寄一封信；
- 刷牙；
- 在头的一侧编辫子；

- 系鞋带；
- 给气球打气。

答案

小测试

(1) – b)

(2) – b)

(3) – b)

避免灾难的发生

炸薯条的做法：

(1) 准备足够的土豆；

(2) 拿一把削皮刀；

(3) 开始削土豆皮，**直到皮都被削干净了**；

(4) **把土豆切成 1 厘米厚的薄片（你不会想把它整个扔下去炸了的）**；

(5) 把平底锅放在炉子上；

(6) 开火并调至中火；

(7) **向平底锅内倒入 100 毫升油（我们可不想一直把瓶子拿在手上）**；

(8) **当油够热了，**将土豆倒入平底锅中；

(9) 炸土豆，**将它们炸至金黄 / 炸 x 分钟**；

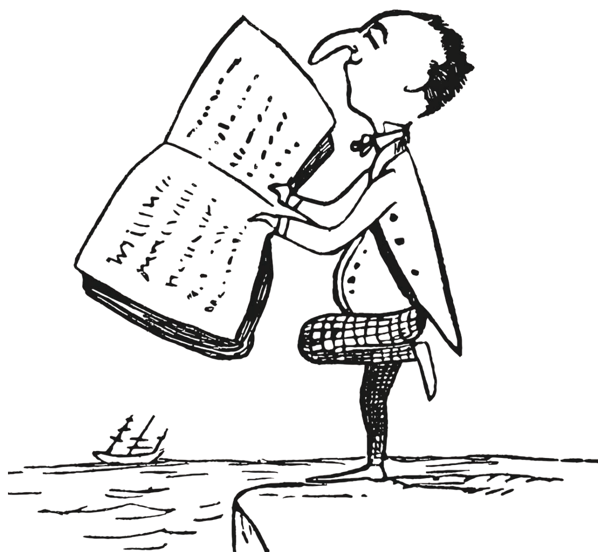
(10) 将土豆从平底锅盛出并放在盘子上；

(11) **关火**。

很明显，这只是一次模拟，并不是真正的程序（虽然我们也可以编写一个厨师机器人的软件）。但你可以看到，确定动作执行的顺序和什么时候停止非常重要，否则有些动作将永远持续下去（之后还会提及这点）。

第 1 章

咒语之书



我们现在要使用一门特别的、专为人类设计的语言编程，通过编程语言写下算法，也就是希望处理器执行的一系列步骤。

但**真正的编程到底是怎样的呢？**我们终于要学习这部分内容了。读完本章，你将有机会编写属于自己的程序。

在本书中，我们基于 Java 语言开展学习。为了在你的电脑上运行 Java 程序，也就是让程序跑起来，你需要下面三件东西。

- 一款用来写代码的**文本编辑器**。它不需要有多特别，免费的文本编辑器就可以，比如本书前面第 X 页向你推荐的那些；
- 一个 Java **编译器**，它能生成所谓的**字节码**；
- 一个**解释器**，能让特定的处理器明白我们编写的代码。

我们可以先把 Java 程序看成是用文本编辑器编写的一系列代码。将它保存为扩展名为 .java 的文件，这样电脑就会把这部分内容当作 Java 代码来处理。

驼　峰



编程世界中有一些规则，规定了如何给程序文件命名。在Java中，这种命名标准被称作**大驼峰式命名法**（UpperCamelCase）。这个称呼生动形象、易于记忆，同时也提醒了我们，.java文件的名字必须是连续完整，没有重音音符、句点或者其他字符的；同时，文件名中的单词之间不能有空格，要首尾相连并大写单词的首字母。这种命名方式方便了编译器的工作，也能帮助我们避免错误。请仔细看，依照这个规则起的名字是不是的确和驼峰的形状很相似。

下面是两个正确的程序命名示范：

```
MyFirstProgram.java
RuleTheWorld.java
```

下面是两个不规范的名字：

`rebelProgram.java`（这不会引起报错，但它没有完全遵循命名规范）

`mariascode.java`（这会给我们带来不便）

让我们来谈几个关键问题。怎样才能编写出有效的命令呢？我可以随便写点什么，然后祈祷电脑理解我吗？是不是用英语就足够了？

这当然不行。跟其他魔法一样，**编程也依赖于前人所著的、写满古老咒语的魔法书和魔法书店，要被正确施加在某个物体上，才能产生特定的魔法。**

下载了第X页提到的JDK后，你不知不觉就拥有了一间Java的魔法书店，编写代码其实就是根据规则调用书中的咒语。要记住，其中有一条规则是：**我们把咒语叫作方法，把魔法书叫作类。**

开始编程时（我保证，再讲几页就真的开始编程了），我们将**调用**希望使用的类，让它们听从我们的命令。接着就可以进一步调用它们的方法，影响我们想要操作的一切。

你好，世界

按照惯例，“你好，世界”（“Hello, World”）是入门编程时最先写下的几个字，我们当然也不例外。

接下来你会看到一个程序示例，这将是我们分析理解的第一段代码。

HelloWorld.java

```
1  public class HelloWorld {
2
3  public static void main (String[] args) {
4
5      // 在屏幕上显示文字
6
7      System.out.println("Hello, World");
8      }
9
10 }
```

注意：在本书中，每当我想要提示你这段内容是 Java 代码时，就会使用这种字体。

首先读到的部分是 **`public class`**。我们把它写在文件的开头，这样编译器在读取时就知道，从这里开始的所有内容都是类（class）。此外，我们还给类起了名字，在这里它就叫“HelloWorld”。

接下来读到的是 **`public static void main...`**，我们总是把它写在主类的开头部分（像个密码一样，可以开启咒语）。

public static void…咒语



这句啰里啰唆、不明所以的话总是出现在程序的主类中。尽管你目前还不需要太在意它，但或许你想知道它的含义。类由以下几部分组成。

public（公共的）。我们有时候会在程序中写一些秘密方法，不允许程序的其他部分读取。但我们又想把这个开启咒语的密码设置成 public，让编译器能够畅通无阻地访问程序。

static（静态的）。虽然将来我们会学习如何创建**对象**（object），但现在我们创建的是类。为了区分类和对象，我们加上了 static 这个关键字。如果什么都不做，我们很可能会把编译器弄糊涂……总之，这是一个非常高级的魔法。

void（空的）。在施加魔法时，有一些方法会有反馈，而另一些方法不返回任何内容。为了避免无谓的苦苦等待，我们必须标记不返回内容的方法，提示这些方法是空的（void）。

main（主要的）。因为 main 的意思是**主要的**，所以我们用它来宣布程序的主要部分。就像田径比赛中的起跑线一样，告诉其他人程序的主体部分从这里开始。

String args（字符串参数）是咒语的一部分，它让程序能够用特殊的顺序运行……你不用太纠结，因为本书暂时还用不到它。但比起不知所以然地盲目重复，我们更需要知道代码运行的深层次原因。

接下来，我们有**一句双斜杠（//）开头的标注**，我们叫它**注释**。这个双斜杠就像是**隐形魔法**！它非常有用，因为如果你把双斜杠加到一句代码前面，程序运行时就不会读取它了。你可以通过注释写下需要记住的事情，解释下一段

代码的作用，而这些都不会**影响**编译器工作。

如果你只写一行注释，可以用双斜杠；但如果你想要写一整段注释，可以为它穿上**隐形铠甲，/* 就像这样 */**，程序就会跳过这部分了。

```
// 这是一条单行注释
/* 这一整段
藏在神奇铠甲下的注释
都将被程序忽略 */
```

最后，我们写下魔法的主体。我们使用一个名为 **System.out.println** 的咒语，它可以**让你的计算机在屏幕上显示内容**。具体来说，就是显示你写在括号里的内容（我们称之为**参数**）。

我们以这个结构来写方法：

```
.方法名("参数文本");
```

也就是说，要这么写：

- 先写一个点（就像用我们的魔杖指明方向一样）；
- 然后写方法名；
- 紧接着在一片迷雾中（好吧，我是指在一对括号和引号中）写我们想要用魔法显现的文字。
- 最后用一个分号来封印以上所有内容。

再来看看这段程序，是不是没那么难懂了？你可以这么理解：打开魔法书（也就是类），宣布我要开始施魔法了（main() 方法），然后就可以开始念咒语了（System.out.println() 方法）。

```
public class HelloWorld {
    public static void main(String[] args) {
        // 在屏幕上显示文本：Hello,World
        System.out.println("Hello, World");
    }
}
```

为了明确那些**包含在其他代码中的代码片段**，我们用这样的花括号 { } 括住它们。千万别忘了使用花括号，否则编译器就会抱怨不知道哪部分到底归谁，也不知道读到哪儿就该停下了。

我们的第一次实践



现在要编写我们的第一个程序了。请非常仔细地阅读以下步骤。

(1) 打开一个文本编辑器，复制我们刚刚分析过的程序。也许你愿意试试先靠记忆默写出代码，之后再查漏补缺。

(2) 保存文件。**和给类命名一样**，别忘了用**大驼峰式**的方法给它命名，并且添加扩展名 .java，关闭文件。

(3) 现在要编译程序。从终端（如果你不知道如何操作，请在“**我们需要准备什么呢**”中寻找相关内容 ）输入 `javac` 命令以及你的程序名和扩展名：

```
javac HelloWorld.java
```

编译器会自动创建一个名为 HelloWorld.class 的新文件，这是我们之后要执行的文件。如果一切进展顺利，你不会看到任何信息提示。

(4) 通过输入 `java` 命令和程序名来运行它——这次不需要扩展名：

```
java HelloWorld
```

成功啦！多亏了魔法咒语，我们的程序成功运行了，终端窗口向全世界显示了我们的第一条信息。

出了点问题？如果在编译时出现了某条错误信息，请检查以下细节：

- 类的名称和文件的名称是否匹配；
- 有没有准确地还原代码；
- 是否加了编译所需的扩展名。

如果第一次没有成功，你也不用太担心。只要仔细一点、耐心一些，重新按照指示走一遍，一定会成功的。

练习

从我们刚刚创建的程序出发，尝试进行以下修改：

- 一个不说“Hello, World”（“你好，世界”）而是说“Bye, bye, crocodile”（“再见，鳄鱼”）的程序；
- 一个先说“Hello, World”（“你好，世界”），然后说“The weather’s good today”（“今天天气不错”）的程序。

答案

- 要修改屏幕上显示的文本，只需改变引号之间的内容即可。千万别忘记打上双引号。
- 要打印两条信息，你可以重复这个方法（即 `System.out.println()`）：尽管这样看起来不太美观，因为它们都挤在一块儿了。

```
System.out.println("Hello, World");
System.out.println("The weather's good today");
```

你也可以把这两条信息放在一起：

```
System.out.println("Hello, World", "The weather's good
                    today");
```

还有其他方法也可以实现这些内容（之后你就会学到了）。

通过这个练习，我们验证了：你可以将需要的所有方法都写在同一个类当中，写在主方法 `main()` 的花括号（`{}`）中，甚至可以向 `System.out.println()` 方法传入若干参数（即双引号中的信息）。

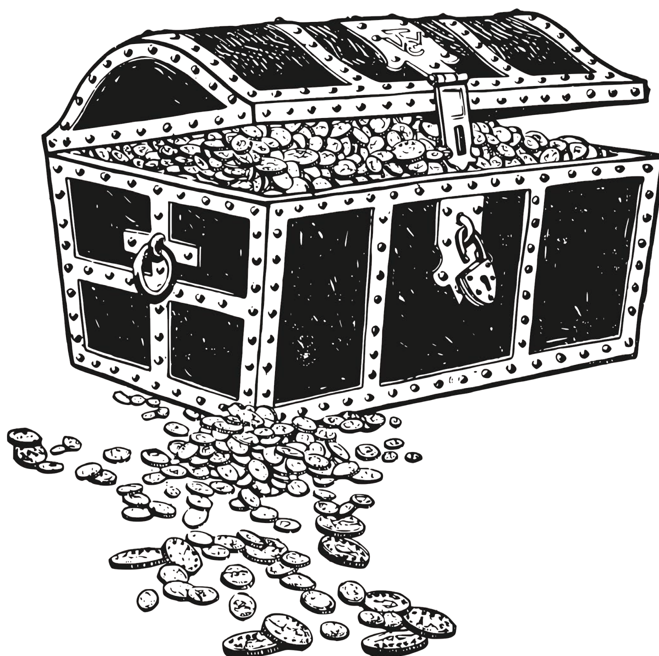
试着实现以下内容：

- 在同一对括号中加入多条带双引号的信息；

- 添加或去除花括号（{}）；
- 添加或去除分号（;）；
- 改变程序名称（就是主方法 `main()` 中提及的名字），让它和文件名称不同。

当你做这些改动时，会发生什么？编译器会反馈给你什么信息？可以成功执行程序吗？

第2章
变量宝藏



我们已经知道了一些程序是如何组成的，以及如何使用编程魔法的知识，是时候学习**变量**这个无比重要的新元素了。

在本章中，我们将学习**什么是变量**、**变量共有几种类型**，以及**如何通过编程掌握它们**。

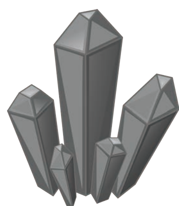
什么是变量

编程教材里写道："**变量是计算机内存中的一个空间，里面存储着我们赋给它的值。**"根据存储信息的种类，变量也分为不同的类型。

这听上去有些不好理解，所以在本章中，我们将像魔法师一样，用神奇的方法创造一座宝藏。

我们的宝藏就藏在一个巨大的箱子里（就像电脑的内存区域一样），里面放着一大堆不同种类、不同形态的数据，它们可值钱了。

宝藏清单上列着不同种类的元素，我们根据场合来使用它们，其中出现频率最高的是下面这些元素。



char（**字符**）：是珍贵的小宝石。背光看看它们的内部，你会发现每个宝石里面都藏着一个带单引号的**字符**（即符号、数字或者字母）。



int（**整数**）：是金币，每一个都有**数值**，这个数可以远小于零，也可以大到无法想象。



double/float（**浮点数**）：金币通常都是**整数**，但有时候我们需要用带小数点的支票付款，比如 25.75 欧元。double（双精度浮点数）和 float（单精度浮点数）便是**有小数点的数字**。



string（**字符串**）：在英语中是绳子的意思。实际上，字符串不过是一串项链，用双引号**把几组字符有序地串起来**，让它们不至于散落到各处。如果它们在百宝箱里散得满处都是，那可就不太好了。



bool（**布尔型**）：本身并不是宝藏，但它们能帮助我们守护这些珍宝。它们是一把把小挂锁，只有两个值：true 或 false（小挂锁的开或关）。或许你现在很疑惑它们有什么用，接下来就会慢慢明白了。

好啦，既然已经知道了这些，让我们言归正传。变量是什么呢？变量是百宝箱中的一个**独一无二**的空间（没错，我们还给这个空间起了名字），**拥有上文提到的某个类型**，并且**包含特定值**。例如，宝藏中的变量可以是：

- 名为 initial（初始）的字符，值为 `'i'`；
- 名为 salary（工资）的整数，值为 `1500`；

- 名为 price（价格）的双精度或单精度浮点数，值为 `1.45`；
- 名为 salute（打招呼）的字符串，值为 `"Hello"`；
- 名为 open（打开）的布尔型，值为 `true`。

在编程时，你可以**按需创建变量**。可以在程序一开始就创建变量（更专业的说法是“声明变量”），一旦程序执行，便会生成变量；也可以在方法中动态地创建变量。即使对变量了解得还不多，你也能够对它进行很多操作。

你可以对整数（int）和浮点数（double）进行数学运算，也可以打乱、截取字符串，还可以使用布尔型让程序做决定……充分发挥你的想象力吧。

创建变量最正确的方式是：变量类型 – 变量名 – 变量值。我们按照这种方式声明并初始化上述变量：

```
char initial = 'i';
int salary = 1500;
float price = 1.45;
String salute = "Hello";
bool open = true;
```

注意：我们把字符的值写在单引号之间，而把字符串的值写在双引号之间。

如果没有提供任何初始值……声明但不初始化变量

当我们不在定义方法的代码中时，我们可以只声明变量，而不用一开始就赋初始值，就像这样：

```
int price;
bool open;
String salute;
```

一旦这样声明了变量，就等同于在内存中创建了一个空间，或者说一块珠宝，并给了它一个名字



（变量名）和一个默认值：如果它是整数，那么默认值为 0；如果是浮点数，默认值是 0.0；如果是布尔型，则是 false；最后，如果是字符或是字符串，那么默认值是“无”，即 null。

如果你在方法中声明变量却不初始化，那么编译时就会报错。

打印变量

让我们回到上一章编写的 Hello， World 程序。我们这一次不告诉程序需要打印的具体内容，而是告诉它变量的名字，让程序展示它的值。

复制这段代码，编译并执行它：

```
1  public class HelloWorld {
2
3  public static void main (String[] args) {
4       char a = "u";
5       int price = 100;
6       String salute = "Hello, World";
7
8      //  在屏幕上显示文本
9
10      System.out.println(u);
11      System.out.println(100);
12      System.out.println(salute);
13
14      System.out.println(price*2);
15      }
16
17  }
```

正如你看到的那样，**先初始化变量，再调用方法并把变量的名字传递给它，方法就知道应该显示什么内容了。**

但是第 14 行代码是怎么回事呢？请仔细观察程序显示的结果。程序把变量 price 的值（100）乘上标示的值，然后显示计算后的结果。为什么它会这样运行呢？

运算符

恭喜你！你刚刚发现了**运算符。这是一些神奇的运算符号，能作用于我们的变量。**

运算符有很多种类型，比如可以操作数值型变量（`int`、`float`、`double` 等）的算术运算符 +、–、*、/ 等，主要负责数学运算。算术运算符**不能用在非数值变量上**（除了 + 符号，它可以用来拼接字符串）。

这些神奇的符号使用起来也十分简单，你只需要了解每个符号的功能。我为你准备了一份使用说明书，你可以在需要的时候进行查阅。请记住，**你不需要背下这个表格**，我们不用像鹦鹉一样机械重复。通过时间的积累和不断的练习，你就能记住这些运算符了。

运　算　符	说　　明
赋值运算符	
=	赋值
算术运算符	
+	加
–	减
*	乘
/	除
%	返回除法运算的余数，例如 7 % 5 = 2
一元运算符	
+	正数
–	负数
++	原值递增 1
––	原值递减 1
!	将布尔变量的值取反（true 变成 false，反之亦然）
相等性与关系运算符	
==	“与……相等”，如 `5 == 5`
!=	“与……不同”，如 `3 != 4`
>	大于
>=	大于等于

（续）

运　算　符	说　　明
相等性与关系运算符	
<	小于
<=	小于等于
条件运算符	
&&	与，例如“周一 && 周二”，表示周一和周二
||	或，例如“周一 || 周二”，表示周一或周二
?:	三元运算符。虽然列在这张表中，但本书并不会用到它
类型比较运算符	
instanceof	“类型是……”，例如“5.6 instanceof Double”的结果是 true

现在让我们来看几个**使用这些运算符的例子**。你有信心猜中这些程序会在屏幕上显示什么吗?

```
1  public class FamilyAge{
2
3  public static void main (String[] args){
4
5          int dadAge = 46;
6          int momAge = 47;
7
8          // 在屏幕上显示文本
9
10         System.out.println("Dad's age is: " + dadAge);
11         System.out.println("Mom's age is: " + momAge);
12
13         int ageAddition = dadAge + momAge;
14
15         System.out.println("Their ages add up to: " + ageAddition + " years.");
16
17         }
18       }
```

```
1  public class FamilyAge{
2
3  public static void main (String[] args){
4
5          int dadAge = 46;
6          int momAge = 47;
7          int sonAge = 10;
8
9          String resultMessage = "Their ages add up to ");
10
11         // 在屏幕上显示文本
12
13         System.out.println("Dad's age is: " + dadAge);
14         System.out.println("Mom's age is: " + momAge);
15         System.out.println("Mom is " + (momAge - dadAge) + " year older than Dad.");
16
17         }
18       }
```

如果编译并执行程序，你会看到我们可以通过运算符 + 和 – 进行加法、减法或是句子的拼接。

此外，正如上面那个初始化变量 `ageAddition` 的例子，我们也能用运算符改变变量的值。在这个例子中，我们用另两个变量值之和来初始化该变量，当然也可以进行其他运算。例如，我们要初始化一个新变量，该变量的值是另两个变量相除后的商：

```
int agesResult = momAge / dadAge;
```

甚至可以用等号：

```
int momAge = dadAge;
```

这意味着初始化变量 `momAge`，使它的值等于 `dadAge` 在赋值时刻的值。

（你可能会有疑问，所以我要解释一下，即使之后改变 `dadAge` 的值，`momAge` 的值也不会发生变化。因为在初始化的时候，`momAge` 已经完成了复制 `dadAge` 变量值的动作，复制后的值和 `dadAge` 的值存在于不同的内存空间。就像我今天拍了一张自拍照，明天换了发型，这张照片也不会因为我的变化而改变。）

因为这些变量属于**相同的类型**，所以我们可以对它们进行等于、相加、相减等运算。如果你试着这样写：

```
String a = dadAge;
```

就会碰到编译错误，这是因为 Java 不允许将存储在整数里的值赋值给字符串。它们不是同一种宝石，因此**并不兼容**。

进击的骆驼



你还记得**大驼峰式命名法**吗？这条规则告诉我们，程序名必须是首尾相连的英语单词，并且每个单词的首字母要大写。

现在来学习另一种规则，它是前一种命名法的变体：**小驼峰式命名法**（lowerCamelCase）。当我们命名变量时，该规则要求使用完整的英语单词，而非缩写。至于字母的大小写问题：如果只使用一个单词，则全部用小写字母表示；如果用两个或两个以上单词，那么除第一个单词外的所有单词首字母大写，例如：

```
int age;
float minimumHeight;
double averageWomenWeight;
```

即使我们不遵从这个规则，编译器也能理解我们的意思，但我们仍应该遵守Java语言的这个标准，因为它能让其他程序员更好地理解我们的代码。

关于变量和运算符的一些挑战



我们将要挑战一些高难度的题目，为此要用上迄今为止还未见过的新方法。

我们要学习 **Scanner** 类（请记住，类就像是一本主题统一的魔法书），它包含了获取用户输入的方法。使用这个强大的方法，我们就可以在运行时**通过键盘和程序对话**。

要使用这个类，首先要把它添加到程序中，告诉程序我们要使用这本魔法书（这个类）了。为此，需要在程序起始处添加以下指示：

```
import java.util.Scanner;
```

小包裹



虽然我们看不见它，但类实际上是属于包（package）的。我们有时候不只导入单个类，还会导入整个包，因为里面包含了我们需要的所有类。

导入了类之后，就可以调用它的方法了。就好像先把书从书架上拿下来，之后就能慢慢阅读需要的咒语了。

要想使用 Scanner 类，我们得遵循以下步骤。

- **打开一个新的 Scanner**，给它起一个名字。**注意**：这是一个叫作 janeDoe 的新 Scanner，从 System.in 中（即键盘）获取用户的输入。用户的键盘（输入来源）作为参数，放在了一对括号中。

  ```
  Scanner janeDoe = new Scanner(System.in);
  ```

- **创建所需类型的变量**。在初始化变量时，我们告诉它，它的值等于我们用键盘输入的内容，就像这样：

  ```
  int i = janeDoe.nextInt();
  ```

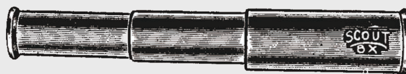
因为我们创建了一个 int 变量，所以程序会等待你输入一个整数（不带小数点的数）。请注意，此处调用方法的形式与之前学习的 println() 方法相同：点 – 方法名 – （参数），在本题中就是：.nextInt()[①]。

一切都准备就绪，我们可以开始输入了。程序会捕获我们用键盘输入的内

① nextInt 不需要参数，所以括号里空的。——译者注

容，并将其存储在内存中，作为一个宝石变量的值。

调用 Scanner 类的其他方法



我们已经学过了如何调用 Scanner 类来读取 int 类型的数据。如果要读取其他类型的数据，只需要**在声明变量的时候更改数据类型**就可以了。

假设我们继续使用名为 janeDoe 的 Scanner 类，让它读取输入的字符串：

```
String myString = janeDoe.nextLine();
```

或者读取布尔型：

```
bool myBoolean = janeDoe.nextBoolean();
```

或者以此类推，读取浮点数（float）、读取一个字符（char）……

但如果我不知道用户会输入什么类型的数据，或者我想让它能够接受任何一种数据呢？好吧，遇到这种情况，你就需要使用条件决策方案了。你将在下节课学习它。

电子存钱罐

我们要设计一个模拟存钱罐的程序。这个程序具有以下功能：

- 接收钱款
- 计算数额
- 显示余额

(1) **编写算法**，叙述使用存钱罐的流程。这里并没有标准答案，许多方法都可以实现这个要求。

(2) 把以下**动作**和可以用来完成这些动作的**方法、工具**连起来：

接收钱款	显示余额	计算金额	输入金额

System.out.println	Scanner	用户的键盘	算术运算符

(3) 一旦完成上述练习，我们就会得到有关程序代码的线索了。

试着写一下代码吧，不能通过编译也没关系。这是一个锻炼构建能力的练习！做完之后再查看参考答案（请记住，答案并不唯一），然后对代码做些必要的改动，直到程序通过编译。

(4) **试验你的新程序**。尝试让它出错。试一下输入小于零的数额，会发生什么呢？

参考方案

(1) 叙述流程

这道题的正确答案并不唯一，以下只作参考：

- 程序开始运行；
- 在屏幕上显示存款总额；
- 询问用户想在存钱罐里存入多少钱；
- 用户输入金额；
- 把输入的金额与存钱罐的存款相加；
- 在屏幕上显示最新的存款余额。

(2) 接收钱款 – `Scanner`

显示余额 – `System.out.println`

计算金额 – 算术运算符

输入金额 – 用户的键盘

(3) 以下是可以成功运行的参考代码（存款初始值可以是 0，也可以是任何其他数额）：

```
1  import java.util.Scanner;
2
3  public class myScanner{
4
5       public static void main(String []args){
6
7           int balance = 0;
8           String neckLaces = null;
9
10          System.out.println("Your balance is " + balance);
11
12          // 询问用户
13          System.out.println("Please enter the amount to deposit");
14
15          // 创建新的 Scanner 对象
16          Scanner myScanner = new Scanner(System.in);
17
18          // 创建用来存储输入金额的变量
19          String myDeposit = myScanner.nextLine();
20          neckLaces = myDeposit;
21
22          // 向用户显示目前的余额
23          System.out.println("Your balance is " + balance + ". Your necklace balance is "
            + necklaces +". Thanks for making a deposit.");
24
25
26  }
27  }
```

如果你运行这段代码，会看到程序依次完成以下内容：

- 显示初始的总金额；
- 询问你输入的金额；
- 等待你输入数额并按下回车键；
- 显示目前的余额。

你也可以用其他方式实现算法。如果感兴趣的话，可以想一想别的方法，并且试着分析哪种实现方式最有用。

(4) **要想让你的存钱罐接收项链（即字符串）或者支票（即浮点数）**，只需要创建其他变量，然后按需使用同一个 Scanner 来接收内容。

试一试以下代码片段（把它添加到程序的主方法中）：

```
// 询问用户
System.out.println(" 请输入存储金额 ")
// 打开新 Scanner 对象
```

```
Scanner myScanner = new Scanner(System.in);
// 创建用来存储输入金额的变量
String myDeposit = myScanner.nextLine();
necklaces = myDeposit;
// 向用户显示目前的余额
System.out.println("您的余额为"+ balance + "。您的项链余额为
                   " + necklaces + "。感谢您使用存款服务。");
```

选做练习：只推荐给进阶水平的学生

你知道怎样修改程序才能让它接受带小数的金额吗？换句话说，除了硬币，你还可以在存钱罐里存下支票。

你知道怎样修改程序才能让它接受字符串变量吗？这样的话，你也就可以把“项链”存到存钱罐里了。

提示：本章后面的附录会有你需要的工具。

附录：你还可以用字符串做其他事

可以看到，字符串（String）是一种和用户“对话”的途径。我们总是希望把字符串装饰得更美观，让程序看上去更专业。现在我要教你一些操作字符串的小技巧，让你创建更复杂的信息，或是处理用户输入的内容。你现在就可以试一试，也可以在贯穿全书的练习中使用这些技巧。

用 `String.length` 计算字符串的长度

```
public static void main(String args[])  {
    String collar = "一条有 76 颗珍珠的项链";
    int length = collar.length();
    System.out.println("字符串长度为：" + length);
}
```

拼接字符串

和之前的做法一样，你可以在一条信息中用加号“+”拼接多个字符串。

同样可以用 String.concat() 方法拼接文本：

```
"我住在".concat("巴塞罗那");
```

或是直接拼接字符串变量：

```
String str_1 = "我住在";
String str_2 = "巴塞罗那";
str_1.concat(str_2);
```

以上两种用法都可行，因为 concat() 方法可以同时接受变量和带引号的文本作为参数。

比较字符串

你可以用 compareTo() 方法比较两个字符串（可以是变量或带引号的文本）是否相同。你会得到一个整数类型变量的比较结果。如果两个字符串相同，结果为 0，否则就会得到一个正数或负数。

示例如下：

```
public static void main(String args[]) {
    String str_1 = "早上好。";
    String str_2 = "晚上好。";
    int result = str_1.compareTo(str_2);
    System.out.println("结果为：" + result);
}
```

（如果使用的是 compareToIgnoreCase() 方法，那么在比较字符串时将忽略大小写。）

替换字符

如果你想隐藏之前编写的内容，可以把某个特定的字母替换成另一个，就像创建了一种密码一样！

```
public static void main(String args[]) {
    String message = new String("My little secret message");
    System.out.println(message.replace('e', 'r'));
}
```

这就可以得到“隐藏”的信息："My littlr srcrrt mrssagr"。

你想加密多少字符都行，然后再根据需要解密。

在 Oracle 的 Java 8 官方指导手册中，你可以找到更多关于字符串操作的建议。

第3章

许多不同的道路



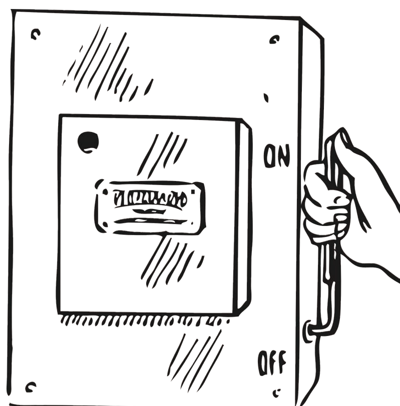
在前面几章中，我们已经学习了编程的基础知识：什么是方法和类，什么是变量，变量有哪些类型，以及如何对变量使用操作符。

不过，一旦开始想要通过编程实现自己的想法，前面学的内容就不够用了。**到目前为止，我们写的程序只懂得一遍遍按部就班地依次执行。**但有时候，我们希望程序能“多想想”。

能不能让存钱罐程序先询问我们想存储什么类型的数值呢？这样它就知道应该创建什么类型的变量了。此外，如果我们输错了数据，程序是否能提醒我们，并且允许我们重新输入呢？

这当然可以实现！（其实我还真没见过有什么是编程不能实现的，只是不知道要怎么实现……）

控制流结构



为了让程序知道在具体情况下应该怎么做，我们需要预判这些情况，并使用所谓的**决策结构**或者控制流。

> **共有 3 种决策结构：**
>
> IF　　IF-ELSE　　SWITCH CASE

为了更好地理解决策结构是如何工作的，我们可以将程序想象成**一个给发光板供电的魔法电路**。电路中布满了电线，其中**一些导线的开关闭合，另一些的则断开**。因此，魔法电流只能流向那些**开关处于闭合状态的灯泡**。只有我们闭合主开关，才会产生电流。电流通过每个开关时，会确认它的开合状态，判断是否能连通开关后面相应的灯泡。如果开关的状态是断开，电流则会绕过这里前往下一个开关。

if 语句

这样是不是一下就理解了。if 语句用来让程序提问，并帮助我们进行比较。要编写这种类型的指令，我们还要使用第 19 页表中列举的条件运算符。

例如，在之前的存钱罐程序中，我们可以增加一条规定：只接受存款不接受取款。要达到这个目的，我们需要在之前进行运算的地方做修改。原来是这么写的：

```
// 创建用来存储输入金额的变量
int deposit = myScanner.nextInt();
    balance = deposit + balance;
```

现在我们可以这么写：

```
// 创建用来存储输入金额的变量
int deposit = myScanner.nextInt();
    if (deposit > 0) {
        balance = deposit + balance;
}
```

注意：使用 >（大于）运算符时，我们发出了这样的指示：如果输入的数值大于 0，就运行花括号之间的代码块，即进行加法运算；如果输入的数值为 0 或是负数，就跳过花括号之间的代码段，读取下一条指令。

因此，正确使用 `if` 语句的方法是：

```
if （待比较的元素）{如果条件满足就会执行的代码块;}
```

你也可以使用已经学过的其他运算符来设立其他规则，例如：

- 只接受数额大于 5 的存款；
- 只有在余额少于 100 时才接受存款（`if (balance<100)`）；
- 如果余额少于 100，就接受存款，但存款后的总额不能超过 100（`if (balance<100 && balance+deposit < 100)`）。

完善电子存钱罐



改写你的存钱罐程序，在其中加入以下新规则：

- 只接受大于 0 的存款；
- 只接受数额为偶数的存款。

你可以使用上面提及的代码片段，不要只是复制或者默写，明白其运行的原理更重要。

如果你已经完成了改写，现在就运行存钱罐，试着往里面输入不支持的金额或内容。

会发生什么呢？

`if-else` 语句

到目前为止，我们已经看到，如果条件满足，就可以通过 `if` 语句让程序执行动作。那如果条件不满足，我们又该怎么办呢？有时候我们希望更进一步，确定以下内容：如果条件满足，程序执行 A 动作；如果条件不满足，那么执行 B 动作。换句话说，我们想要一个**只有条件不满足时才进入的语句块**，我们称之为 `else` 语句块。

这和上个例子中的照明电路一样：使用 `if` 语句，我们可以发出“如果开关闭合，那就点亮灯泡”的指令。而**使用 `if-else` 语句，我们就可以发出“如果开关闭合，那就点亮灯泡，否则就拉响警报”的指令**。因此，如果灯泡亮了的话，警报就永远不会响，因为只有 `if` 语句的条件不满足时，`else` 中的语句块才会执行。

下面来看一个 Java 代码的实例：

```
int freezerTemp = 3;
    if (freezerTemp >= 4) {
    System.out.println("冰箱内部的温度过高！ "+ "温度为"
                        + freezerTemp + "摄氏度。");
    } else {
    System.out.println("冰箱正常工作中。");
}
```

这是一段控制冰箱的程序代码。可以看到，程序首先会将温度值存储到一个变量中。接着，它会评判温度是否超过了 3 摄氏度。如果高于 3 摄氏度，程序会提醒我们冰箱出了些问题；如果温度低于 3 摄氏度（也就是在其他情况下），程序会反馈给我们另一条消息，确认冰箱正常工作中。

试运行这段代码，在这个例子中你将会看到第二条消息。如果你把温度值改成 4 摄氏度或以上，显示的则会是第一条消息。

这条句法很简单，你只需要**把新的代码块加到 `if` 语句块之后就行了**：

条件

如果条件 == true，执行这条语句块

否则，执行这条语句块

也就是说：

```
if ( 待评估的条件 ) {
    动作 1; } else {
    动作 2;
}
```

每个代码块都有属于自己的花括号 { }，每个动作也有属于自己的分号 ;。

除此之外，我们还可以使用这个语句完成更多操作，比如**添加任意多的代码块！**这很简单，**只需要把第二位至倒数第二的代码块命名为 else if，并加上它们待评估的条件**。例如，下面这段代码会指出我们在考试中是刚好通过、取得了优异成绩还是不及格：

```
public class NailOrFail {
    public static void main(String[] args) {
    int examGrade = 8;
    if (examGrade >= 8) {
        System.out.println("恭喜你以优异成绩通过考试。");
    } else if (examGrade >= 5) {
        System.out.println("你通过了考试。");
    } else {
        System.out.println("很抱歉，你未能通过此次考试。");
    }
    }
}
```

注意：虽然变量 `examGrade` 的值同时满足了前两个条件（单从逻辑上讲，如果大于 8，那么也一定超过 5），一旦程序将第一种情况评估为真，那么它将执行对应的代码块，并放弃接下来所有的 `if-else` 语句块。程序**不会继续**检查是否还满足其他条件。

还有一个重点：请注意编写条件的方式。如果只写“大于 8”“大于 5”等类似的条件，那么当你正好是 8 分或 5 分时，程序会告诉你没有通过考试。这样的话多可惜啊！

再次改进存钱罐



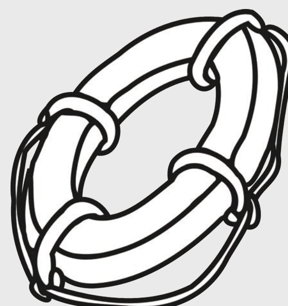
看完上面的例子，你可以再次改进存钱罐程序。如果在已经实现的 `if` 语句中添加一个 `else`，那么当用户试图存入一笔不符合要求的存款时，程序会显示一条消息，提示因为操作有误存款失败。

在进行操作时，没人想看到计算机毫无反应，也不告诉我们发生了什么。所以我们要养成尝试预见错误的好习惯，在出现问题时对用户的行为进行反馈。

你的改进有没有效果呢？如果没有，可以参考后面一页的答案。

防御性编程！

防御性编程是设计程序的一种方式，即使是用户的操作错误，它也会尽可能地保证程序良好运作。这种编程方式非常有用，它不仅让我们的应用程序更可靠，还使它不那么轻易就受到黑客的攻击。



虽然就你目前所学，这种设计原理有点超前，但你的确可以开始使用其中的一些理论来检查自己的程序了：

- 编写尽可能**简洁**的代码；
- 请小伙伴们帮忙**检查**代码；
- **测试**你的程序（进行错误操作、输入类型错误的数据等）；
- **重复使用**写好且正常运行的代码（就像对存钱罐程序做的那样）。

不管是现在还是将来，如果你有兴趣深入了解相关内容，都可以参考网上关于防御性编程的条目。

参考方案

接下来，你会看到解决前面那个挑战的参考方案。

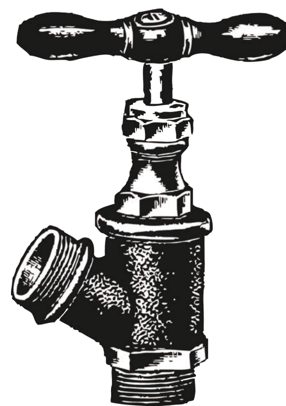
我们还可以完成其他操作，比如把这个句子：

```
System.out.println("你目前的金额为 " + balance);
```

放到第 18 行之后。那么只有当金额发生改变时，程序才会显示它的值。请记住，编程可以用无数种方法实现一种算法——只要你们的小脑瓜能想得到。

```
1  public class MyPiggyBank{
2      public static void main(String[] args){
3
4              int balance = 0;
5
6              System.out.println("Total balance is " + balance);
7
8              // 询问用户
9              System.out.println("Please enter the amount to deposit");
10
11             // 创建新的 Scanner 对象
12             Scanner myScanner = new Scanner(System.in);
13
14             // 创建用来存储输入金额的变量
15             int deposit = myScanner.nextInt();
16
17             if (deposit > 0){
18                 balance = balance+deposit;
19              } else {
20                 System.out.println("Deposit rejected. You must enter an amount over zero.");
21              }
22
23              // 显示目前的余额
24              System.out.println("Total balance is " + balance);
25
26         }
27     }
```

switch 语句



除了 if 和 if-else 语句，我们还有第三种可以控制代码执行的工具：switch 语句。

switch 在英语中就有开关、断路器的意思，而该语句的作用也正是如此：**切断所有与初始条件不一致的指令**。就像是在一个管道网络中，按照我们的意愿打开阀门，让水流流过指定管道。switch 语句基于一系列可能的路径，而程序只会选择那些符合条件的道路。

为了更好地理解如何使用这个语句，让我们一起来看下面的例子。这是先前存钱罐程序的答案，不过这里使用 switch 语句替代了 if-else 语句：

```
// 创建用来存储输入金额的变量
int deposit = myScanner.nextInt();
    switch (deposit) {
    case 0: System.out.println("拒绝存款请求：存款金额必须大于0。");
            break;
    case 1: balance = balance + deposit;
            break;
    case 2: balance = balance + deposit;
            break;
    case 3: balance = balance + deposit;
            break;
    default:
            break;
    }
    // 显示目前的余额
    System.out.println("您的余额为： " + balance);
    }
}
```

让我们一步步来看。

- 第一步先编写 switch 语句并注明**变量名，该变量作为参数**用来和每种情况（case）做比较。
- 在花括号中**列出每个待比较的情况**，此处我们列出了存款为 0、1、2、3 的情况。
- 每个情况都包含**待执行的指令**。
- **break 语句紧接在每个情况之后**，用来提示程序离开 switch 语句块。也就是说，如果符合情况“0”，那么程序就不会读取其他情况的指令。
- **default（默认）**情况是可选项，它表示“如果变量值和所有情况都不符，那么执行这条默认情况”。

你可以使用没有默认情况或默认情况为空的 switch 语句。为了证明这一点，试着用双斜杠 // 注释掉那几行代码，查看程序是否能通过编译。

注意：这里使用了类型为 int 的变量来进行比较。switch 语句还可以用来比较 char、int 甚至 String（如果你使用的是 Java 7 或更高版本）。一定要记得检查待比较变量的类型，它必须和你在每种情况中注明的数值类型是相同的。

正如你看到的那样，在这种情况下用户可能输入任何数字，因此再使用 switch 语句就不那么合适了。为了应对所有可能，我们必须为每一个可以放入变量中的整数（–3、–2、–1、0、1、2、26……）编写一种情况，但这根本做不完。

那我们怎么知道什么时候用哪种语句呢？

- 如果我们想要比较**数据区间**（数值大于或小于），那么用 if 或 if/else 语句更好。
- 如果我们想要比较一些**特定的情况**，或是寻找精准的匹配，那么 switch 语句更好，因为基于这个语句编写的代码会更清晰、更易读，运行时间也更短。

保护我们的存钱罐



为了防止未得到许可的好奇小伙伴偷偷使用存钱罐，我们将使用新学到的语句来保护它。

在启动存钱罐时，使用你已经学过的 `switch` 语句和 `Scanner` 方法，要求用户输入密码。如果密码匹配，他就可以继续查看余额；如果密码错误，程序会向用户发送消息告诉他密码不正确，所以没有权限查看余额。

(1) 编写存钱罐的算法，包括启动、请求密码、验证密码等步骤。完成这一步很重要，它能帮助我们明确设计步骤和之后要编写的内容。

(2) 尝试编写代码，添加新功能。请记住，应该把新语句块放在计算存款额的部分之前，而且要提前保存密码以便之后进行比较验证。

参考方案

我还得再重申一遍：没有标准答案，以下只是本题无数可行方案中的一种。

```
import java.util.Scanner;

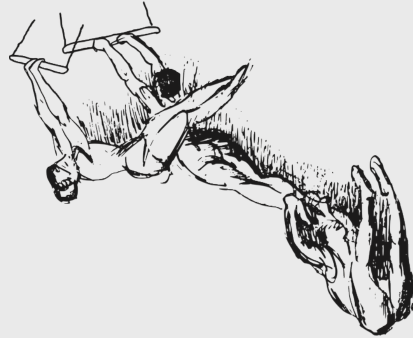
public class MyBank {
    public static void main(String[] args) {
        String password;
        int balance = 0;
        String askForPass = "请输入您的密码并按下回车键";
        String passwordOK = ":::: 欢迎进入您的安心电子存钱罐。:::";
        String total = " >>> 您的余额为 ";
        String thanks = "::: 感谢您使用电子存钱罐。:::";
        String KO = "密码错误。您无权使用该电子存钱罐。";
            // 创建新的 Scanner
            Scanner myScanner = new Scanner(System.in );
            System.out.println(askForPass); // 询问密码
            // 创建一个字符串变量用于存储用户输入的密码
```

```
                // 根据已存储密码（abracadabra）验证
                password = myScanner.next();
                switch (password) {
                case "abracadabra":
                System.out.println(passwordOK + "\n" + total + " " + balance);
                // 询问存款金额
                System.out.println("请输入存款金额");
                // 创建一个 int 变量存储存款金额
                int deposit = myScanner.nextInt();
                if (deposit > 0) {
                balance = balance + deposit;
                } else {
                System.out.println("拒绝存款请求：存款金额必须大于 0。");
                }
                // 显示余额
                System.out.println(total + balance + "\n" + thanks);
                break;
                default:
                System.out.println(KO);
                break;
            }
        }
}
```

这种解决方案有一些需要考虑的重要细节。

- 我们把显示余额以及计算存款额的语句放到了 switch 语句块中的第一个 case 里，因为我们希望只有用户输入了正确密码，才运行该部分程序。
- 我们在 default（默认情况）关键词下创建了一条消息，如果用户输入的密码不正确，程序会显示此消息提醒他。

再来点更难的！



在输入密码进入电子存钱罐时，密码直接显示在了屏幕上，你觉不觉得有点美中不足？这意味着任何从你身后走过的人都能看见密码，那真是太糟糕了……

Java 中有一个名为 Console.readPassword() 的方法，用它来获取写入的密码（类似于 Scanner 读取器），就不会在屏幕上直接显示输入的内容了。

和使用 Scanner 类的时候一样，要调用这个方法，你需要导入（import）它的类：

```
import java.io.Console;
```

接着，你可以删除或注释掉把密码存进变量的那句代码，并换成如下代码：

```
Console cnsl = null;
cnsl = System.console();
char[] passString = cnsl.readPassword();
String pass = new String(passString );
```

可以看到，使用它与使用 Scanner 读取器的方法十分相似。你需要创建一个新的控制台（我们称之为 cnsl，当然你也可以给它起个别的名字），命令它从系统控制台读取内容，并创建一个变量（passString）用来保存输入的内容。

这样一来，你的程序就相当专业了。

第 4 章

孩子们的游戏



到目前为止，我们已经可以对程序进行一些控制了：我们知道了如何创建变量，如何使用运算符管理它们，如何用方法改变变量，甚至根据条件来决定执行不同的任务。

可是，我们的程序仍然缺点什么。

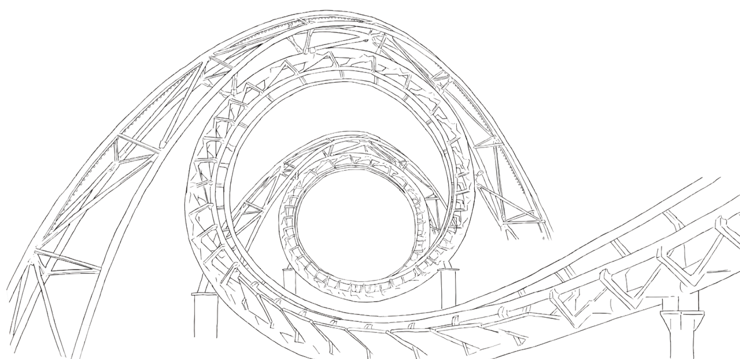
在存钱罐的例子中，如果你第一次输错了密码，程序就会马上关闭，不给你再次尝试的机会——这显然有点奇怪。有时我们只是不小心输错了，肯定还希望再输入一次。**在不重复编写代码块的前提下，我们该怎么告诉电脑要多次执行同一个任务呢?**

刚好有一帮叫作循环（loop）的助手可以帮我们解决这个问题，它们可能干了！就像前一章的条件语句一样，**循环也是控制程序动作流的工具**。两者的区别在于，条件语句用来说明要干什么，而**循环则说明一件事情要执行几次或持续多久**。

什么是循环

字典里写道，英语单词 loop 有“卷发”的意思。我们也用 loop 这个词表示过山车轨道上一个个翻转的圈式弯道。一般来说，我们把围绕中心旋转的东西叫作“loop”。

编程中的循环（loop）指的是**一个不断重复的指令块，直到某个条件满足才会停止。**



所以在存钱罐的例子中，我们可以通过构建循环，给用户更多的机会输入正确密码。告诉程序，每次密码有误时，重复**读取键盘输入 – 比较输入密码和正确密码 – 拒绝或授予权限**的行动序列。循环次数是由我们来决定的，可以确定一个具体的输入次数，也可以允许用户无限制地尝试直到输入正确的密码为止。

Java 语言里有三种循环，我们都会在本章中学习。

WHILE–DO WHILE–FOR

`while`

while 在英语中是“当……时”的意思，而我们在编程中用 `while` 语句来

评估一个条件的值。当这个值为真（true，即符合条件时）时，**程序才会执行 while 语句中的代码**。

(1) 程序**先检查**条件的值是否为真。

(2) 如果值为真，那么就**执行 while** 语句中的代码。

(3) **再次检查**条件的值。

(4) 如果仍然为真，就**再次运行代码块**。

一旦在检查中发现**条件的值为假（false）**，程序就会停止执行该代码块，并继续执行程序的其余部分（换句话说，**它将退出循环**）。

请记住：

- 如果在第一步就发现条件的值为假，那程序就不会执行这段代码，**而且再也不会重新检查这个条件的值**——它会直接跳向下一段代码；
- 如果条件的值永远为真，就会产生一个**无限循环**，也就是说这段代码会一直运行下去。

你听说过网球发球机吗？这是一种能不断抛出网球的机器，网球运动员通过它进行训练，只要发球机不停下，他就必须击球。这个例子很好地解释了 while 循环的概念。在这里，条件的内容是“只要发球机继续抛出网球”，而待执行代码的内容则是“用球拍把球击回去”。当该条件为真时，执行击球的代码块；当发球机不再发球时，停止击球，退出循环。



我们是这样编写 while 循环的：

```
while （条件） {
 代码指令；
}
```

就是这么简单！

下面这个程序模拟了火箭发射倒计时的场景：

```
public class RocketLaunch {

    public static void main(String args[]) {
        int countDown = 5;
        String message = "IGNITION";

        while (countDown > 0) {

            System.out.println(countDown);
            countDown--;
        }
        System.out.println(message);
    }
}
```

一定要注意这三点。

- 这里的条件是倒计时变量（countDown）**大于** 0。如果我们写 >=，程序将会在屏幕上多显示一次倒计时变量的值。在你的计算机上验证一下这两个符号的差别吧。
- 语句 countDown--; 表示“countDown 变量的值减 1”。因为我们把这句话写入了循环，所以**只有在执行代码块的时候**，也就是每次在屏幕上显示 countDown 的值之后，**变量的值才会减** 1。
- 我们将打印消息的代码放在了循环**外面**。因为如果将它放进循环，每当屏幕显示数字，就会打印一次消息——这意味着我们的火箭会升空五次！

练习



我建议你通过编写下面这几个程序来练习 while 循环。

(1) 观察下面的代码片段。你觉得屏幕上会显示什么呢？

```
int i = 0;
while (i < 3) {
    System.out.println("echo");
```

```
    i++;
}
```

(2) 下面这段程序肯定出了点问题，它的循环一经运行就无法自动停止，必须强制关闭。你知道问题在哪儿吗？

```
int counter = 0;
while (counter <= 3) {
    System.out.println("Message to show");
}
counter++;
```

(3) 给进阶学生的拓展练习。

给定一个数字（由用户输入），编写程序计算这个数能被 2 整除几次，并显示结果。

你可以这么编写算法：

- 程序要求用户输入数字；
- 用户输入数字 6；
- 程序使用 **`while`** 循环将数字除以 2，再将结果除以 2，以此类推，直到结果不能被 2 整除为止（6/2 可以整除，3/2 为 1.5，即不能被 2 整除，所以结果为 **1 次**）；
- 屏幕上显示结果："6 可以被 2 整除 1 次"。

答案

(1) 单词会显示 3 次：

```
echo
echo
echo
```

(2) 因为我们把语句 `counter++;` 写在了 `while` 语句块的外面，所以这段代码会无限循环。程序只有退出循环，变量值才会递增，因此在这段代码中，

不管循环多少次，counter 的值都不会增加。

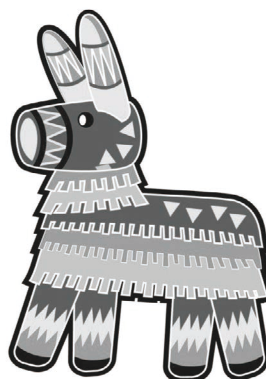
(3) 我解这道题的思路是这样的：只要除法的余数为 0 就继续循环。判断是否整除可以通过比较 enteredNumber - enteredNumber/2*2 是否等于 0。这里使用了 Java 的小技巧：整数除以整数的结果始终为整数，如果不能整除，则对结果进行截取，只取整数部分。如 5/2 的数学结果为 2.5，根据 Java 的整数除法规则，截取整数部分，因此 Java 中的结果为 2。此时，如果对 5/2 的结果再乘以 2，结果为 4，不等于 5，因此可以判断 5 不能被 2 整除。反之，如果 enteredNumber 能被 2 整除，则先除 2 再乘 2 的结果必然等于 enteredNumber。

```
import java.util.Scanner;

public class CountDivision {
    public static void main(String args[]) {
        int counter = 0;
        Scanner scanner = new Scanner(System.in);
        int enteredNumber = scanner.nextInt();
        while (enteredNumber - enteredNumber/2*2 == 0) {
            enteredNumber = enteredNumber/2;
            counter++;
        }
    System.out.println("It can be divided " + counter + " times.");
    }
}
```

另一种方法是：开始循环，通过取模（module，类似于取余）运算符判断用户输入的数字能否被 2 整除；如果成立，则继续循环。

do-while



do-while 语句与之前的 while 语句很相似，区别是 do-while 语句**先执行代码块再检查条件的值**。

这就像皮纳塔游戏[①]：

- 用棍棒击打皮纳塔；
- 检查一下，皮纳塔破了吗？
- 如果还没有，那么继续击打；
- 再次检查有没有打破。

以此类推，当变量 `intactPinata` 的值为假（`false`）时，我们才退出循环，不再继续击打皮纳塔了。

正如你看到的，待执行的动作（用棍棒击打）在检查条件前就已经执行完毕了。换句话说，不管发生什么，**动作至少会执行一次**，所以我们把这个语句叫作 `do-while`。

简单的挑战



让我们回到存钱罐程序。尝试加入 `do-while` 语句，实现一个新功能：只要用户没有输入正确的密码，程序就会不断请求用户再次输入。

在着手编写前先好好思考一下。我建议你先用铅笔打个草稿，至少写一写计划实现的算法。

为了更好地完成练习，我有一些建议：

- 请记住，你要先询问一次密码，然后再检查用户的输入是否正确；
- 密码的比较结果将成为评估条件；
- 如果密码输入正确，则结束循环，允许用户进行存款；
- 如果密码输入错误，则打印信息提示错误，并再次请求输入密码。

① 皮纳塔是墨西哥一种独特的节日装饰，用纸糊的容器装满糖果和玩具，悬挂起来，可以用棍棒击打，打破时糖果和玩具会掉落下来。常见的外形为小驴子。——译者注

解决方案

```
import java.util.Scanner;
public class MyPiggyBank {

    public static void main(String[] args) {
        int balance = 0;
        String pass;

        // 创建新的 Scanner 类
        Scanner myScanner = new Scanner(System.in);

        do {
            // 请求用户输入密码
        System.out.println("请输入您的密码并按下回车键");
            // 创建用来存储用户输入密码的变量
            pass = myScanner.nextLine();
        } while (!"abracadabra".equals(pass)); // 如果密码正
            确，不再请求输入

        System.out.println(":::: 欢迎进入您的安心电子存钱罐。:::
            \n >>> 您的余额为 " + balance);
        // 询问存款金额
        System.out.println("请输入存款金额");
        // 创建用来存储输入金额的变量
        int deposit = myScanner.nextInt();
        if (deposit > 0) {
            balance = balance + deposit;
        } else {
            System.out.println("存款金额必须大于 0：存款请求被拒绝");
        }
        // 显示最新余额
        System.out.println("您现在的余额为 " + balance +
            "\n::: 感谢您使用安心电子存钱罐。:::");
    }
}
```

for

`for` 语句用于编写一类特定的循环：**当我们事先知道所需的迭代（循环代码块）次数时。**

要编写这个语句，首先需要定义两个变量，一个拥有初始数值，另一个是要达到的最大值。每次执行该代码块时，第一个变量数值将增加 1。当第一个变量值增加至与第二个变量值相等时，迭代结束。

这就好比在捉迷藏游戏中，约翰要寻找他的 6 个小伙伴。



- 变量 `foundKids`（已找到的小伙伴）的初始值为 0。
- 变量 `hidingKids`（藏起来的小伙伴）值为 6。
- 现在约翰开始玩游戏了。因为变量 `foundKids` 的值是 0，他绕着院子开始寻找，先找到了玛利亚。
- 现在 `foundKids` 的值变为 1。但它还不等于 `hidingKids` 的值，所以约翰必须再绕一圈。
- 约翰又绕着院子转了一圈，然后找到了迈克。
- 现在 `foundKids` 的值变为 2。这个值仍然没有达到 6，这意味着约翰还得再找一圈。
- 就这样进行下去，`foundKids` 的值总会增加到和 `hidingKids` 的值相同：都是 6。这时约翰就退出循环，不再寻找了。

如果用 Java 代码来呈现以上内容的话，是这样的：

```
class MyHideAndSeek {
    public static void main(String[] args) {
        int hidingKids = 6;

        for (int foundKids = 0; foundKids < hidingKids;
            foundKids++) {
            System.out.println("约翰又绕着院子转了一圈，已经找到
                了" + foundKids + "个小伙伴。");
        }
    }
}
```

请注意以下几点。

- `for` 语句的基础参数有 3 个，依次为**初始**、**结束**和**增量**，即循环变量的初始值、结束值以及每次循环增加的值。
- 初始值 0 也计入一次循环。计算循环次数时，永远不要习惯性地从 1 开始数。如果你想循环 6 次，（初始值为 0 时）就应该把结束值设为 5。
- 循环的初始参数是一个变量，应该在 `for` 循环内部初始化。如果你在程序的一开始就进行循环初始化，会导致程序无法通过编译，因为变量超出了循环的范围——这点我们之后还会提及。
- 循环的结束参数总是一个算术比较式（比较大小的表达式）。当这个表达式为假时，循环停止。
- 循环变量值的增量发生在每轮循环后，而不是循环前。
- `hidingKids > foundKids` 非常重要。如果你写的是 `>=`，那要等到 `foundKids` 的值为 7 时算术比较式的结果才为假，那么约翰就会多绕一圈。**只有在还没找齐 6 个小伙伴时才需要继续进行循环，这点非常重要。**
- 我们在循环的花括号 {} 里放入了要重复执行的代码，这和 `if` 语句的编写形式相同。

三个有关 for 循环的练习



(1) 尝试编写一个程序，使它显示 0~50 的所有奇数。

小提示：我建议你回头看看那张算术运算符的表格，里面有一个运算符正好能用在这里。

(2) 尝试编写一个程序，使它显示 0 到某个用户输入数字之间的所有奇数。

(3) 修改第 (2) 个练习的程序，使程序在显示所有奇数后，再显示一条消息，告诉我们一共找到了多少个奇数。

请记住：这条消息必须在程序执行完毕后显示。

解决方案

(1) 既然我们知道循环变量的最大值是 50，而且这个值不会发生变化，那么你只需要编写一个执行 50 遍的循环，让每次循环都对 i 的值（初始值为 1）进行评估，检验其是否被 2 除后余数为 0。

```
1  import java.util.*;
2  public class OddCounter{
3      public static void main(String[] args){
4
5      for (int i = 1; i < 50; i++){
6          if (i%2 != 0) {
7              System.out.println(i);
8              }
9          }
10     }
11 }
```

(2) 要完成这个练习，我们只需加上一个面向用户的提问并对 for 循环稍加修改。现在，循环的结束值不再是一个固定不变的最大数，程序拥有了一个存储用户输入数字的变量。

```
1  import java.util.*;
2  public class OddCounter{
3  
4      public static void main(String[] args){
5  
6      Scanner myScanner = new Scanner(System.in);
7      System.out.println("请输入一个整数");
8      int num = miScanner.nextInt();
9  
10     for (int i = 1; i <= num; i++){
11         if (i%2 != 0) {
12             System.out.println(i);
13             }
14         }
15     }
16 }
```

注意：在 for 循环的代码中，我们写的是 <=。因为如果你写了 i < num 而用户输入了 5，那么循环在 i 等于 4 的时候就会停下，根本不会打印最后一个奇数 5。选择哪种比较操作取决于你要完成的任务：可能要求你筛选两个数之间（包含这两个数）的所有奇数，也可能是求除首尾两个数之外的所有奇数。

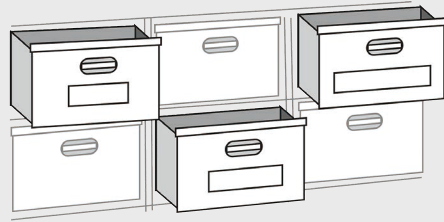
(3) 在这种情况下，我们只需要完成以下步骤：

- 创建一个变量，作为奇数的计数器；
- 每当 i 为奇数、进入 if 语句块时，计数器加 1；
- 退出 for 循环后打印消息。

```
1  import java.util.*;
2  public class OddCounter{
3  
4      public static void main(String[] args){
5      int counter=0;
6  
7      Scanner myScanner = new Scanner(System.in);
8      System.out.println(" 请输入一个整数 ");
9      int num = miScanner.nextInt();
10 
11     for (int i = 1; i <= num; i++){
12         if (i%2 != 0) {
13             System.out.println(i);
14             counter++;
15             }
16         }
17         System.out.println("0 和 "+num+" 之间一共有 "+counter+" 个奇数。");
18     }
19 }
```

数组：另一种保存数据的方式

Java 有一种特殊的数据结构，我们可以用它**对一组类型相同的变量进行归类**。你可以把数组想象成一组抽屉柜，每个抽屉中都存放了一个值。



例如，可以这样声明一个名为 daysOfTheWeek 的**字符串数组**：

```
String[] daysOfTheWeek = new String[7];
```

这行代码表示，我想创建一个名为 daysOfTheWeek 的字符串类型数组，其中有 7 个抽屉（请记住，我们总是从 0 开始计数）。

为了填满抽屉，使用如下语法：

```
daysOfTheWeek[0] = "Monday";
daysOfTheWeek[1] = "Tuesday";
```

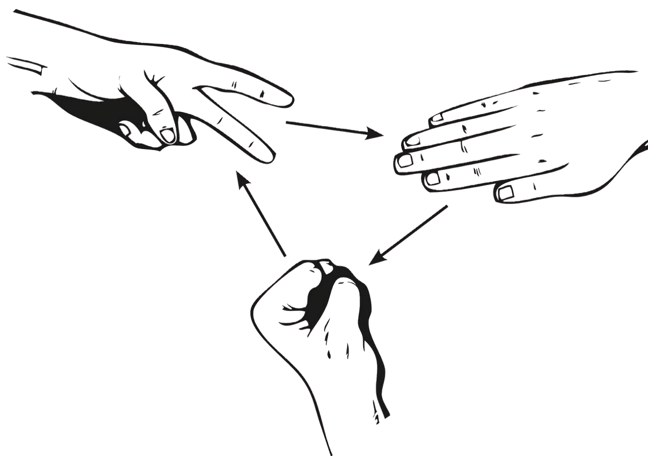
其中，数字表示你保存数据的位置。在这里例子中，我们把字符串 "Monday" 放在了 0 号抽屉中。

使用 for 或 while 这样的结构来遍历（即把数组中的每个数都访问一遍）和操作数组非常常见：塞满抽屉，清空它们，看看里面都有什么，将里面的内容与其他变量进行比较……在循环遍历时，我们通过数字来引用存放数据的抽屉，这些数字被称为数组的索引。

本书不会深入研究这个数据结构，因为我觉得你在学习了面向对象的编程后再深入了解这部分内容，会更加有收获。

尽管如此，了解这个知识点还是很有必要的。如果你想提前了解剧情的话，在本书的最后一个练习中，你将有机会第一次使用数组 。

游戏：石头剪刀布



之前我们已经为编程做了充分准备，接下来，带上已经掌握的内容和马上要学习的小技巧，来创建第一个属于我们的互动游戏吧！

和所有优秀的专业人士一样，在开始敲击键盘之前，我们必须先完成几个预备步骤。

- 首先，我们要整理**需求**。换句话说，我们要写下游戏规则和程序应当具备的功能。一般来说，每条需求之后都会被转换成特定函数，由编程来实现。
- 接下来要设计算法，也就是执行程序时的步骤顺序。
- 然后，就可以开始编写代码了。
- 再下一步是测试程序，验证它是否按预期运行。
- 最后进行重构：修改代码使之更简洁、更专业，尽可能获得最好的结果。

让我们赶紧开工吧！

程序需求分析

没有人生下来就什么都会。因为这是你第一次编写一款真正的游戏，所以我将列出一份程序需求清单。在接下来的练习中，我建议你先试着自己思考，然后再验证你想的是否和参考答案一样，有没有遗漏什么重点。

游戏必须具备的功能包括：

- 启动游戏的同时显示一个菜单；
- 菜单里有“开始游戏”和“退出游戏”两个选项；
- 游戏中的对手是程序与用户；
- 程序向用户展示出拳指令——0 是布，1 是剪刀，2 是石头；
- 在每轮游戏中，用户使用键盘出拳，而程序随机出拳；
- 剪刀赢布，石头赢剪刀，布赢石头；
- 如果是平局，程序将告知用户；
- 如果有一方胜出，会显示一条信息宣布胜出的玩家；
- 程序能预见用户可能随时输入的不正确数据，并避免由此产生的错误；
- 在第 2 版程序中改进游戏流程，如果用户没有选择退出则自动进入下一轮。

程序的算法

现在你需要发挥想象力，在脑海中重演一遍完整的游戏顺序。与此同时，以事件列表的形式写下每一个步骤，以便之后形成算法。

在下一页，你将看到我的参考算法。但在此之前，先试着不看答案自己完成一下。

(1) 显示开始画面，提供选项：1) 开始游戏，2) 退出游戏。

(2) 如果用户选择 2)，则关闭程序。

(3) 如果用户选择 1)，则显示游戏指南：0 是布，1 是剪刀，2 是石头。

(4) 如果用户输入了除最初两个选项外的其他内容，则显示“选择错误”的信息，并回到第 (1) 步。

(5) 用户出拳，游戏回合开始。

(6) 程序随机出拳。

(7) 比较双方出拳结果：

a. 如果程序和用户出拳一样 ，则显示“DRAW!”（“平局！”）；

b. 如果程序战胜了用户，则显示“YOU LOSE!”（“你输了！”）；

c. 如果用户打败了程序，则显示“YOU WIN!”（“你赢了！”）；

d. 如果用户的输入值无效，则显示“INCORRECT MOVE”（“出拳错误”），游戏返回第 (3) 步。

动手写下以后，是不是发现算法没有之前想象的那么难了？有条理地组织内容能帮助我们更好地处理逻辑问题。

编写代码

要编写代码的话，我的建议是逐步实现若干功能，每完成一个就进行检验，确保其正常运行。

为了便于编写，我将待实现的功能划分为几块。你需要创建一个新类，并尝试自己完成这些步骤。之后，你可以参考我的代码。（我在代码中添加了一些注释，这样你会更清楚每段代码对应的步骤。）

1. **编写主菜单**

编写一个显示在屏幕上的菜单，提示用户可选内容（“开始游戏”或“退出游戏”）。

2. **编写菜单的逻辑**

根据用户的选择执行：1) 开始游戏，2) 退出程序，3) 报错并重回第 (1) 步。

建议：你可以使用 `System.exit(0)` 命令退出程序。当用户输入除 1) 和 2) 之外的内容时，我建议你使用某种决策结构或者循环语句，防止错误发生。

3. **编写游戏回合的逻辑**

逻辑顺序依次为：

- 采集用户的输入信息；
- 让程序随机出拳；
- 比较用户和程序的出拳；
- 宣布比赛结果：平局 – 程序胜 – 用户胜 – 错误输入。

提示：在 Java 语言的 `Math` 类中，有一个方法能从预先提供的范围里挑选随机数，它可以帮助你实现随机出拳。

你必须导入 `Math` 类才能使用这个方法（就像你导入 `Scanner` 类一样：`import java.util.Math`）。下面的示例就使用了这个方法，完成的内容和练习的要求也非常相似。

```
import java.util.Random;
public class random {

    public static void main(String[] args) {
        // 创建一个 random 对象
        Random random = new Random();

        // 选择 0~100 的一个整数，并把它保存在 randomInt 变量中
        int randomInt = random.nextInt(100);

        // 在屏幕上显示这个整数
        System.out.println(randomInt);
    }
}
```

在开始编程前，我是这样搭建程序框架的：

```
import java.util*;

public class RockPaperScissors{

    public static void main(String[] args){

    /* TODO：编写开始菜单
      编写一个显示在屏幕上的菜单，提示用户可选内容（"开始游戏"或"退出游戏"）*/



    /* TODO：编写菜单的逻辑。采集用户的输入，以及：
    1）开始游戏
    2）退出程序
    3）报错并重回第 1）步 */
    /* TODO：编写游戏回合的逻辑
    比较用户和程序的出拳
    宣布比赛结果：平局 / 程序胜 / 用户胜 / 错误输入 */

  }
}
```

TODO（待办事项）？

在英语中，“To do”的字面意思是“要去做……”。在代码注释的开头使用“TODO”是一个被广泛认可的惯例，表明程序在这块区域还有需要完成的工作。



解决方案

下面是同样一段代码，但使用的是我已经准备好的类，代码部分已经补全。我把代码分成了几部分，这样你可以更好地看清细节。

1. 主菜单

```
1   import java.util*;
2
3   public class RockPaperScissors{
4
5       public static void main(String[] args){
6
7       /* TODO：编写开始菜单。
8        编写一个显示在屏幕上的菜单，提示用户可选内容（"开始游戏"或"退出游戏"）*/
9        System.out.println("::::::::::::::::::::::::::\n"
10                       +"::ROCK, PAPER, SCISSORS::\n"
11                       +"::        WELCOME!      ::\n"
12                       +"::::::::::::::::::::::::::\n"
13                       +"Choose an option:\n 1) PLAY  2) EXIT ");
14
```

可以看到，只要稍微修饰一下文本，就能让程序看起来更像一款游戏。如果你已经完成了这一步，建议你检查一下程序能否按预期编译并运行。

2. 菜单的逻辑

```
15      /* TODO：编写菜单的逻辑。采集用户的输入，以及：
16      1）开始游戏
17      2）退出程序
18      3）报错并重回第 1）步 */
19      Scanner scan = new Scanner(System.in);
20      int option = scan.nextInt();
21      // 如果用户输入错误，这个循环会请求用户再次输入
22      while (option!=1 && option!=2){
23              System.out.println("Please choose one of the valid options:\n"+
24                                 " 1 - PLAY  2 - EXIT");
25              option = scan.nextInt();
26      }
27      // 一旦得到有效的用户选项，这个 switch 语句将负责管理选项
28      switch(option){
29          case 1:
30      /* TODO：编写游戏回合的逻辑
31      比较用户和程序的出拳
32      宣布结果：平局 / 程序胜 / 用户胜 / 错误输入 */
33            break;
34          case 2: System.exit(0);
35            break;
36      }
37    }
38  }
```

在编写菜单逻辑时，我们使用了一个 `while` 语句。在用户提交有效选择之前，`while` 语句会反复请求他重新输入。

一旦用户输入了正确的选项，就会退出循环，然后根据选项进行下一步：

- 如果选择了 1，那么游戏回合开始；
- 如果选择了 2，将调用 `System.exit(0)` 方法退出程序。

3. 游戏回合的逻辑

下面的内容只是**单轮**游戏回合的逻辑，我们还没有到开发多轮游戏的层次。需求分析部分也提到过，多轮游戏将在程序的第 2 版中实现。

因此，我们现在只需要把单轮游戏的逻辑放入 switch 语句。

```
28    switch(option){
29    /* TODO：编写单轮游戏的逻辑
30    比较用户和程序的出拳
31    宣布结果：平局 / 程序胜 / 用户胜 / 错误输入 */
32            case 1: Sistem.out.println("Choose your move:\n"+
33                "0 - PAPER 1 - SCISSORS - 2 - ROCK");
34                int userMove = scan.nextInt();
35            }
36            // 让程序随机出拳
37            Random random = new Random();
38            int machineMove = random.nextInt(2);
39            // 显示双方出拳情况
40            System.out.println(" 你出的拳是 "+ userMove +"。 程序出的拳是 "+ machineMove
41                        +"\n::::::::::::::::::::::::::::::::::::::::::::::\n");
42            // 比较双方出拳并选出优胜者
43            if(machineMove==0 && userMove==0){
44              System.out.println("DRAW!");
45            } else if (machineMove==0 %% userMove==1){
46              System.out.println("YOU WIN!");
47            } else if (machineMove==0 && userMove==2){
48              System.out.println("YOU LOSE!");
49            } else if (machineMove==1 && userMove==0){
50              System.out.println("YOU LOSE!");
51            } else if (machineMove==1 && userMove==1){
52              System.out.println("DRAW!");
53            } else if (machineMove==1 && userMove==2){
54              System.out.println("YOU WIN!");
55            } else if (machineMove==2 && userMove==0){
56              System.out.println("YOU WIN!");
57            } else if (machineMove==2 && userMove==1){
58              System.out.println("YOU LOSE!");
59            } else if (machineMove==2 && userMove==2){
60              System.out.println("DRAW!");
61            }
62          break;
63
64        case 2: System.exit(0);
65          break;
```

我要再次说明，能实现以上功能的编程方式不止一种。为了能让你看清语句的执行顺序，理解每行代码的作用，我选择了最浅显易懂的方式来编写这个功能。

这部分代码的实现仅仅基于我们已经学习掌握的工具，虽然很基础，但也相当稳定可靠。

程序的第 1 个版本已经准备就绪，现在可以进入测试阶段了。

测试程序

在开发新程序的过程中，为了保证产品质量，我们会进行各种操作。

这些操作的目的是，模拟使用该程序时可能发生的所有情况，并检验在每种情况下，程序是否都能正确响应。接下来，我们会以测试的形式编写所有的情况，然后执行这些测试进行检验。如果程序没能对其中某些场景正确响应，我们就要重新编写代码来解决以上问题。测试人员可以基于不同的因素开展测试，比如程序的需求、他先前的经验、之前已经发生的错误，等等。

我建议用运行程序，用玩游戏的方式进行测试，检查程序是否存在错误，或是否如预期正常运行：

- 主菜单：选择退出；
- 主菜单：选择一个错误选项；
- 主菜单：选择开始游戏；
- 第一轮游戏：选择剪刀；
- 第二轮游戏：选择布；
- 第三轮游戏：选择石头；
- 第四轮游戏：选择一个错误选项。

一切正常吗？如果没有的话，再检查一遍你的代码，把它和我的参考答案比较一下。

重构游戏

重构代码是一个修改程序结构的过程。在改进代码的同时，不影响程序原有的功能。

也就是说，**它并不是要改变程序所做的事，而是尝试用更简单、更清晰、更稳定的代码实现相同的效果。**

重构代码是一个非常好的习惯，它能帮助我们改进工作，督促我们不断学习。

如何改善我们的游戏?

如果你留意的话，会发现目前的代码结构有些臃肿。

我们有好几个循环，甚至还有一些嵌套结构（比如 `switch` 语句里还有 `if` 语句和 `while` 语句），这让代码**读起来有些费力，也迫使我们必须小心翼翼地调整代码，因为嵌套语句这么多，修改一处小细节就会影响其他所有内容。**

在**整理**代码时，我们可以做的第一件事就是改进 `if` 结构。请注意看代码，如果我们使用 `if(machineMove==userMove)` 这个语句，而不是分别比较 0 等于 0、1 等于 1、2 等于 2，就可以省去三行代码。

我们还重复了好几次相同的字符串（宣布平局、你赢了、你输了），其实我们可以在程序开始的地方用 3 个变量定义这些字符串。

通过这两个简单的修改，`if` 语句块现在是这样的：

```
// 比较双方出拳并选出优胜者
if (machineMove == userMove) {
      System.out.println(draw);
      } else if (machineMove == 0 && userMove == 1) {
      System.out.println(userWins);
      } else if (machineMove == 0 && userMove == 2) {
      System.out.println(machineWins);
      } else if (machineMove == 1 && userMove == 0) {
      System.out.println(machineWins);
      } else if (machineMove == 1 && userMove == 2) {
      System.out.println(userWins);
      } else if (machineMove == 2 && userMove == 0) {
      System.out.println(userWins);
      } else if (machineMove == 2 && userMove == 1) {
      System.out.println(machineWins);
      }
      break;
```

```
case 2:
      System.exit(0);
      break;
```

除了能够简化代码，这样做还有一个好处：如果将来我们想修改输赢时显示的消息，你只需要在声明变量的位置进行修改即可。

此外，把一些代码片段**封装**起来整合成方法，也能有效地改进编程。

除了直接使用包含在 Java 库里的方法（比如我们已经学习的 Scanner() 方法和 Random() 方法），你也可以创建属于你的自定义方法，将其放在单独的代码区域中，在需要的时候调用它们。

我们要进行以下操作，用管理游戏回合的代码创建一个方法。

(1) 在 main() 方法的外面创建一个新方法（你不能在一个方法内部创建另一个方法）。想知道哪里是方法外面的话，可以参考区分代码块的花括号 { }。

```
public class MyClass {
    public static void main(String[] args) {
        // main 方法内部的区域
    }
    /* 在类内部，但在 main 方法外面的区域。你可以在此放置 round 或你编写
的其他方法。*/
}
```

(2) 像前几章那样，按访问修饰符（如 private、public）、返回类型、方法名、参数以及花括号 { } 的顺序声明方法：

```
public static void round(){}
```

(3) 我们把执行整个游戏回合的代码从 case 1 移动到 round() 方法里，然后在 case 1 处调用新方法。我强烈建议你注释掉（加上 //）switch 语句的代码，不要剪切粘贴，而是将它复制进新方法。一旦你发现哪里出了问

题，打算放弃先前的修改，这样做不会让你丢失已经编写好的、正常工作的代码。

(4) 程序的整体结构如下所示（为了代码的可读性，我特意删去了 TODO 注释）：

```
import java.util.* ;
public class RockPaperScissors {
    public static void main(String[] args) {
    // 主菜单
    System.out.println("::::::::::::::::::::::::::::\n"
                      + ":::      石头、剪刀、布     :::\n"
                      + ":::          欢迎！          :::\n"
                      + "::::::::::::::::::::::::::\n\n"
                      + "请选择：\n 1 - 开始游戏 2
                                 - 退出游戏 ");
    // 菜单的逻辑
    Scanner scan = new Scanner(System. in );
    int option = scan.nextInt();
    // 如果用户输入错误，则请求他再次输入
    while (option != 1 && option != 2) {
        System.out.print("请输入有效选项：\n" + " 1 - 开始游戏
            2 - 退出游戏");
        option = scan.nextInt();
    }
    // 一旦得到有效的用户选项，这个 switch 语句将负责管理选项
    switch (option) {
    case 1: round();
        break;
    case 2: System.exit(0);
        break;
    }
}
```

注意：我已经把一部分代码删掉了，并替换成对 `round()` 方法的调用。因为创建的 `round()` 方法运作时不需要任何参数，所以 `round()` 方法的括号内是空的——是的，绝对是空的，连空格都不能放！

(5) 把代码移动到方法中。注意，相应地增减花括号。

```
    /* 游戏回合的逻辑 */
    public static void round(){
        System.out.print("选择你的出拳：\n"+
                         "0 - 布 1 - 剪刀 2 - 石头");
        Scanner round = new Scanner(System.in);
        int userMove = round.nextInt();
        // 这个循环检查用户的输入是否正确
        while (userMove!=0 && userMove!=1 && userMove!=2) {
            System.out.print("选择你的出拳：\n"+"0 - 布 1 - 剪
                刀 2 - 石头");
            userMove = round.nextInt();
        }
        // 让程序随机出拳
        Random random = new Random();
        int machineMove = random.nextInt(2);
        // 宣布用户和程序的出拳情况
        System.out.println("你出的拳是 "+ userMove +"。 程序出
               的拳是 "+ machineMove
               +"\n::::::::::::::::::::::::::::::::::::\n");
        String draw = "平局！ ";String userWin = "你赢了！ ";
        String machineWin = "你输了！ ";
        // 比较双方出拳并选出优胜者
        if(machineMove==userMove){
            System.out.println(draw);
        } else if (machineMove==0 && userMove==1){
            System.out.println(userWin);
        } else if (machineMove==0 && userMove==2){
            System.out.println(machineWin);
        } else if (machineMove==1 && userMove==0){
            System.out.println(machineWin);
        } else if (machineMove==1 && userMove==2){
            System.out.println(userWin);
        } else if (machineMove==2 && userMove==0){
            System.out.println(userWin);
        } else if (machineMove==2 && userMove==1){
            System.out.println(machineWin);
        }
    }
}
```

现在我们迎来了最终的测试：保存代码并检查是否通过编译。如果编译不成功，你需要检查自己的代码和本书的示例有什么不同。

精益求精

最后的最后还有一件事非常重要：我们要把负责特定功能（如管理游戏回合）的代码分离出来，这样你就可以在不改变主方法（main()）的情况下，修改或扩展这段独立代码，当然也可以按需反复使用它。

你还记不记得，我们在制订游戏需求时曾提到，希望能在程序的第 2 版中实现多轮游戏，这样就不用必须重启才能玩下一轮了。

在实现这个功能时，分离代码的另一个作用就体现出来了。

我们在同样的区域（即在类里面、main() 方法的外面）创建一个**新方法**，方法名为 menu()（菜单）：

```
public static void menu() {
    Scanner scan = new Scanner(System.in);
    int option = scan.nextInt();
    // 这个循环检查用户的输入是否正确
    while (option != 1 && option != 2) {}
    System.out.println("请输入有效的选项:\n" + " 1 - 开始游戏
        2 - 退出游戏");
    option = scan.nextInt();
    // 一旦得到有效的用户选项，这条 switch 语句将负责管理选项
    switch (option) {
        case 1:
            round();
            System.out.println("你想再玩一局吗? 1- 是 2- 否");
            menu();
            break;
        case 2:
            System.exit(0);
            break;
    }
}
```

在 menu() 方法的 case 1 语句中，调用 round() 方法的下一行，我们打印了一条消息，询问用户是不是还想继续游戏。紧接着，再一次调用 menu() 方法。

如果你仔细看，就会明白这段代码的意义：当你在菜单中选择进行游戏时，程序会先开启新一轮游戏，然后返回菜单。这样一来，每当用户选“1) 开始游戏”想继续玩下去时，我们就把游戏回合串联起来啦。

挑战成功啦！

第 5 章

寻踪觅迹



到目前为止，我们编写的所有程序在运行时都不会在文件系统中留下痕迹。

在这一章，我们将学习编写能和文件系统交互的程序，**包括搜索本地文件、创建新文件和使用文件保存修改**。这些功能对编程很重要，我们可以让电子存钱罐记录不同时期的余额，或是在电子游戏中存档，记录获得胜利的瞬间。

有些关于文件的概念你得先知道

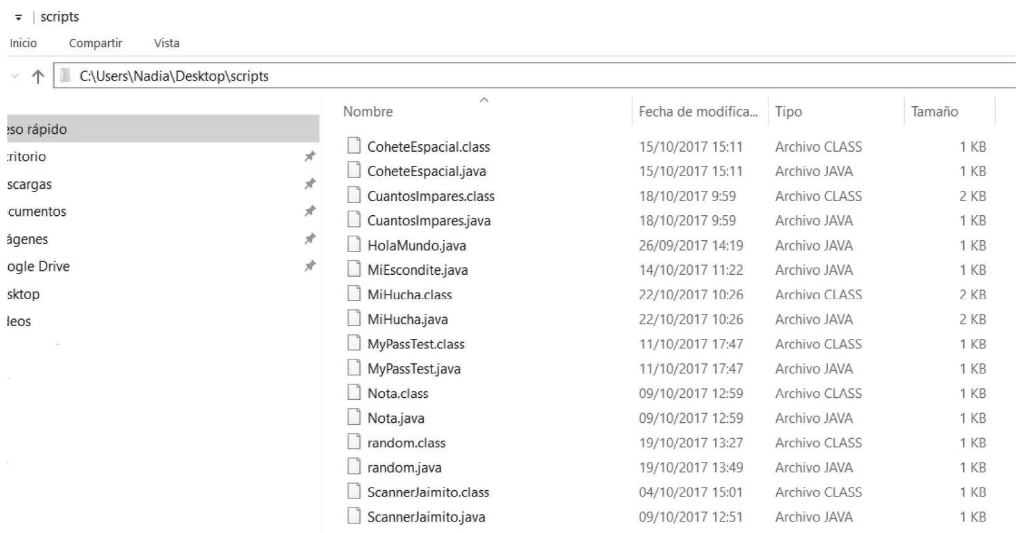
我们已经掌握了变量的概念，而文件和它相似，也是**计算机存储中划分出来的一个小空间，具有特定的内容和格式**。用更专业的话来说，文件是被分配了名字的位集，并且含有其所在文件夹的说明。

在文件夹系统中，有一串名称标记了文件存放的确切位置，我们称之为**路径**。

比如说，有一个以 .class 为后缀的文件，内容是前一个练习中完成的游戏

程序，我给它起名为 RockPaperScissors.class，这就是它的文件名。我将它直接保存在计算机桌面上，于是这个文件的路径就是 C:\Users\Nadia\Desktop\RockPaperScissors.class。

试一试：随便打开电脑上的一个文件夹（比如下载文件夹、程序文件夹或者存放了文档的文件夹）。观察一下，地址栏中指向这一文件夹的路径都是怎么显示的。



不同的操作系统（Windows、Ubuntu 和 MacOS 等）对路径的显示稍微有些差异，但大体是相同的。

通过尝试和观察，我们知道了路径就是**计算机存储中指向文件的地址**。

路径一共有两种类型。

绝对路径：前一个例子中显示的就是绝对路径，这种路径**由代表存储单元的字母或名称开始**。也就是说，它显示了从根目录到文件的完整地址。

相对路径：相对路径**只显示我们所在位置到文件的地址**。拿前面提到的这个路径 C:\Users\Nadia\Desktop\Scripts\program.class 为例，如果用户或程序已经位于系统中的 /Nadia 文件夹下，那么相对路径就表示为 /Desktop/Scripts/program.class。

在编写程序的时候，如果我们需要和文件路径打交道，一定要弄清楚需要写的是相对路径还是绝对路径。

File 类



File（文件）是一个用来管理文件的 Java 类。更准确地说，它管理的是文件位置。需要强调的是，尽管 File 类的名字会让人误以为它负责管理文件本身，但它实际上管理的是存放文件的路径。

掌握 File 类提供的一系列方法后，你就能实现许多有关文件的操作，其中包括：

- 验证一个文件是否存在；
- 创建文件夹；
- 重命名、移动、复制或删除文件。

注意：我们**无法**通过 File 类直接读取文件中的内容，这点之后还会提到。

调用 File 类的方法与调用 Scanner 类和 Random 类的方法相似（但是和其他两个类不同，File 不是方法的集合，而是一种数据类型）。

(1) 声明 File 变量之前，在程序起始处先导入 java.io.File。

(2) 实例化 File 类：File file = new File(String filePath);。

(3) 和之前一样，你可以为变量起任何名字（不一定叫 file）。你可以直接传入文件的地址字符串作为参数，或者像我一样，传入包含文件地址的字符串变量。

让我来总结一下：在实例化 File 类时，我可以传入包含文件地址的字符串变量，创建一个新的 File 对象：

```
String notesPath = "C:/Documents/Homework/Notes/thursdaynotes.doc";
File myNotes = new File(notesPath);
```

也可以直接用文件路径作为参数：

```
File myNotes = new File("C:/Documentos/Deberes/Apuntes/
apuntesJueves.doc");
```

我更推荐使用变量，因为它能方便我们维护代码。如果文件的路径发生了改变，我并不需要进入方法重复修改，只需要更改变量本身。另外，每次程序启动时，我还可以询问用户希望的路径，把它存入变量中。

File 类的一些实用方法

下面介绍一些非常简单实用的方法，你可以用它们来管理文件。

.exists() 用于检查文件是否存在。如果某个文件不存在，调用依赖于这个文件的方法就会报错。所以，先运行 .exists() 方法可以预防这类错误发生。该方法会生成一个布尔值。如果文件存在，值为 true；如果文件不存在，则值为 false。

```
File file = new File(path);
bool fileExists = file.exists();
System.out.println("文件在那儿吗？: "+ fileExists);
```

.mkdir()（或 mkdirs()）用于创建一个新文件夹，并返回一个布尔值。如果文件夹成功创建，值为 true；如果创建失败，则值为 false。

```
File file = new File(path);
bool createDir = file.mkdir();
System.out.println("文件创建成功了吗？: "+ fileExists);
```

.renameTo() 除了能改变文件名，还可以改变文件路径——该方法会把文件移动到新的路径。也就是说，.renameTo() 并不会更改文件名，而是会寻找我们指定的新路径。如果路径存在，方法会将文件移动到新路径。同样，这个方法在完成操作后也会返回一个布尔值。如果移动成功，值为 true，否则值为 false。

```
File a =  new File("C:/User/Desktop/Maria.jpg");
File b = new File("C:/User/MyPics/Maria.jpg");
bool movesFile = a.renameTo(b);
System.out.println(" 你移动文件了吗？ "+ movesFile);
```

可以看到，在这个例子中，文件名并没有发生改变（都是 Maria.jpg），改变的是文件的路径。我们**移动**了文件 Maria.jpg 的位置，现在它位于一个叫 MyPics 的文件夹内。

`.delete()` 用来删除当前文件，并生成一个布尔值：如果删除文件成功，值为 `true`，否则值为 `false`。

```
File file = new File("C:/Users/Ana/Desktop/test.txt");
boolean delete = file.delete();
System.out.println(delete);
```

注意：使用 `.delete()` 后，文件不会被移动到回收站，而是会被永久删除，所以测试这个方法时千万要小心谨慎。

try/catch

在管理文件的时候，Java 编译器会强制我们使用一个特别的语句块，所以在继续学习之前，我们应该先了解一下它。

在执行之前练习编写的程序时，你有可能会遇到程序异常。换句话说，**在程序运行过程中，遇到了非同寻常和出乎意料的情况，而此时程序并不知道如何处理**，由此产生的错误就叫异常。

例如，在电子存钱罐的第 1 版程序中，当用户试图输入的内容不是整数而是字符串时，程序就会毫无准备、不知所措。如果你尝试输入字符串的话，将会得到如下结果：

```
Exception in thread "main" java.util.InputMismatchException
    at java.util.Scanner.throwFor(Unknown Source)
    at java.util.Scanner.next(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at java.util.Scanner.nextInt(Unknown Source)
    at MiHucha.main(MiHucha.java:26)
```

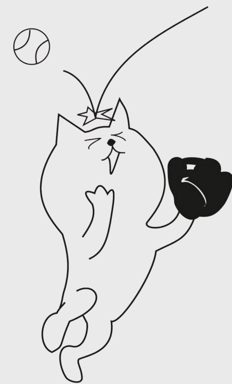
与此同时，程序将停止运行。

这就是异常。学习这个概念需要有一定的编程基础，所以我们并不会深入讲解，但你应该了解一些基础知识。异常有 checked 和 unchecked 两种类型，即检查型异常和非检查型异常。当发生检查型异常时，编译器会强制我们预防这类异常的发生。预防的措施就是使用 `try/catch`（尝试 / 捕捉）语句块。

总结：**不管什么时候，只要使用了会产生检查型异常的方法，就必须加上 `try/catch` 语句块。否则代码写得再完美，都不会通过编译。**

异常发生时，程序到底怎么了？

我猜你一定对这个问题很好奇。



有时即使程序通过了编译，执行过程中也会发生**意想不到的事件**（比如没有给方法的参数赋值、输入数据的类型不匹配，等等）。

这时候，发生异常情况的方法会创建一个对象，其中**包含异常错误的相关信息**：为什么会发生错误、事件发生时系统处在什么状态……我们把这一过程叫作**抛出异常**。

异常就像是一个被抛出的球，从产生异常的方法传递到更高层次的方法。触发传递时，程序会检查这些方法中是否存在处理异常的代码。如果没找到这样的方法，这个叫“异常”的球就会继续从一个方法向上传递到另一个方法，直到传递到主方法。如果还是没找到管理异常的“负责人”，程序

就会突然结束。最后，你会在控制台看到一条信息，**告诉我们有一个异常抵达了主方法，以及它的类型是什么**。

有一些异常的名称一目了然，你马上就明白发生了什么情况。

InputMismatch：输入数据的类型不匹配。
FileNotFound：在指定路径无法找到某个文件。
NoSuchMethod：在程序的某个地方调用了不存在的方法。

通常，你还可以看到是哪一行代码引起了错误：

at MyPiggyBank.main(MyPiggyBank.java:26)

处理异常的语句块由三部分组成，分别为 try、catch 和 finally（即尝试、捕捉、最后）。使用方法如下所示：

```
try {
 // 有一些方法强制我们捕捉和处理异常，我们把包含这些方法的代码放在这里
} catch （异常类型 异常名）{
 // 处理异常的代码放这里；只有当发生异常时，代码才会运行
} finally {
 // 不管有没有异常，放在这里的代码都会运行
}
```

我们用这个语句块组织代码，告诉程序**尝试**（try）做某件事。如果无法完成，在异常传递到其他方法前先**捕捉**（catch）异常，**最后**（finally）再执行别的动作。

finally 语句是可选的；也就是说，即使不加入这句话，也能顺利通过编译。不过还是很有必要学习一下 finally 语句，因为它通常被用来清除或关闭进程。比如，我们要在 try 语句块里**打开**一个文件查看内容，这时使用 finally 语句块来关闭文件就是个好习惯。不论 try 语句块是否抛出异常，在程序继续执行其他任务或结束运行前，finally 语句块都会正确地关闭文件。

还有一点特别重要：在异常发生时，程序一般会停止运行。我们要确保，即使运行失败，对其他部分造成的影响也能降到最低。这就是编程中经常提到的“以最优雅的方式运行失败”。

> 我们曾经提到，finally语句块的代码**一定**会被执行。但如果较真起来的话，这句话**并不完全正确**。
>
> - 如果try或catch语句块中有System.close(0)命令，那么程序永远不会执行finally语句块。
> - 如果finally语句块里发生了错误，程序也不会执行其中的代码，而是会抛出另一个异常。
>
> 看，编程里永远都有例外。

使用 try/catch/finally 的示例

为了有助理解，接下来我要在之前学习过的火箭发射倒计时示例中运用这个语句块。我们选择这段代码作为示范是因为它比较浅显易懂，虽然这个例子本身并不强制要求使用try/catch（因为没有调用任何会抛出**检查型**异常的方法）。

```
public class RocketLaunchNew {
    public static void main(String args[]) {
        int countDown = 5;
        String message = "IGNITION"; // 点燃
        try {
            while (countDown > 0) {
                System.out.println(countDown);
                countDown--;
            }
            System.out.println(message);
        } catch (Exception e) {
            System.out.println("出现了一个异常：" + e);
        }
    }
}
```

注意：我给**异常起了个名字**，因为我觉得把异常（exception）叫作 e 比较直观。我还在 `catch` 语句块中设置了一条消息，用来显示发生的异常。程序会描述异常发生的具体情况，而不是简单地把字母 e 打印出来。

还有其他许多可以管理异常的方法，但如果只是要让程序通过编译，掌握上述内容已经完全足够了。

现在，让我们回到本章的主题，继续学习管理文件。

再谈 Scanner 类

第 21 页介绍了 `Scanner` 类读取信息或数据流的方法，也完成了一些相关练习，比如如何使用 `nextInt()` 和 `nextLine()` 读取用户用键盘输入的数据。

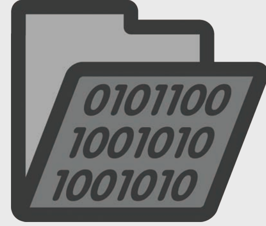
其实，**`Scanner` 类还可以用于读取来自系统文件的数据流**。就像之前读取由键盘输入的字符串一样，通过调用合适的方法，`Scanner` 类可以对文件（如 doc 文件）里的文本字符串进行管理和操作。



实际上，文件中的数据都是逐行**按顺序排列**的。当你想访问并处理这些数据时，程序的读取顺序和它们在文件中排列的顺序基本是一样的，所以我们称之为**数据流**。你可以把数据想象成水流，读取的时候，它们就一个接一个地从水龙头里流出来。

接下来，我们将学习如何使用 `Scanner` 类提供的方法读取文件的字符流。

字节流与字符流



字符（char）是一种对应单个字符（比如一个字母）的数据类型，而字节（byte）则是信息的基本单位。

尽管这两者看上去差不多，但实际差别可大了。它们不仅占用的空间不一样（一个字符的大小可以是一个字节的两倍），用途也不同：字节用来存储二进制数据，而字符则用来存储字母。

因此，**面向字节的文件**包含的是字节信息，所有的字节相互紧挨着排列；而**面向字符的文件**包含的是文本字符串，由空格或换行符等隔开。比如，一封用文本编辑器写的信就是面向字符的文件。

希望通过上面这段简短、基础的解释，你能清楚区分所用的文件是面向字符的还是面向字节的。

我们使用 `File` 类告诉 `Scanner` 类希望打开的文件，并用路径定位文件。

和读取键盘输入内容的方法类似：

```
Scanner keyScan = new Scanner(system.in);
```

我们通过以下方式读取文件：

```
Scanner fileScan = new Scanner(File path);
```

请注意，使用 `Scanner` 类的方法和之前一样，唯一的区别是，作为参数的不是键盘输入而是文件路径。

无论是从文件读取还是从键盘读取，我们在 `Scanner` 类中都调用相同的方法：`nextInt()`、`nextLine()` 或任何其他适合文件数据的方法。

示例

在系统中新建一个文档，文档内包含下面这句话。

```
I have read the whole text.
```

（我建议你将其保存为 .txt 后缀文档。）

现在复制这段代码。要注意文件路径，它必须和电脑上文件的实际路径一致。

```
import java.io.File;
import java.util.Scanner;
public class FileReader {
    public static void main(String[] args) {
        try {
             // 设置路径
             File path = new File("C:/Users/Desktop/text.txt");
             // 打开 Scanner 类，传入路径作为参数
             Scanner fileScan = new Scanner(path);
             // 让 Scanner 类读取文件的第一行，也就是读到第一个换行符停下
             String firstLine = fileScan.nextLine();
             // 在控制台屏幕上显示读取的内容
             System.out.println(firstLine);
             fileScan.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

注意：当我处理完一个文件后，我**一定一定一定**会使用这个语句关闭文件：`fileScan.close();`。这一点和饭后必须刷牙一样重要。

如果复制代码的过程没有问题，你会看到控制台上显示的正是文件里的那句话。

太棒啦！现在我们知道如何在程序中读取文档啦！

让我们来做一些实验



(1) 如果把上面程序中的 `nextLine()` 改成 `next()` 方法，会发生什么？为什么会这样呢？

(2) 把 `next()` 方法替换成 `nextInt()` 方法又会发生什么呢？

(3) 再试试把 `String firstLine = fileScan.nextLine();` 这行代码换成下面的代码块：

```
for (int i=0; i<4; i++){
      String firstLine = fileScan.next();
      // 在屏幕上显示任何读取到的内容
      System.out.println(firstLine);
}
```

你能够得出什么结论呢？

解决方案

(1) 把 `nextLine()` 换成 `next()` 时，程序只会读取句子的第一个单词。这是因为方法的工作方式不同：`nextLine()` 读取下一行文本，而 `next()` 读取的是下一个空格前的文本，也就是下一个单词。

(2) 这个操作会导致程序抛出异常。因为 `nextInt()` 方法原本期待的输入内

容是由空格或换行符分隔的整数，但现在却遇到了字符串，所以会发生异常。因为我们用 `catch` 语句块控制了异常，所以能够处理错误并打印发生错误的原因。

(3) 这个小实验展示的是，如何通过决策和循环结构读取文档的一部分。这个编程思路可以帮助我们实现许多功能，比如：

- 只读取包含某些词语的文档；
- 读取一篇文档的其中一部分；
- 一边读取文档，一边处理它的内容；
- 以及任何你想实现的功能。

`PrintStream` 类

我们已经学会了如何查找并读取文件，下一步该学习如何写入文件了。为此，我们会使用一个新的类，叫作 `PrintStream`（字面意思是**打印数据流**）。

尽管这是我们第一次和这个类打交道，但有个好消息要告诉你，`PrintStream` 的主要方法对我们来说一点都不陌生，就是 `println()` 和 `print()`。**`print()`** 只是把作为参数的字符串写入文件，而 **`println()`** 在写入字符串后还会加上换行符。

注意：

- **如果你需要在字符串之间添加空格，必须自己手动加入**，否则每次写入字符串时内容都会挤成一堆；
- **如果你在已经存在的文件里使用这些方法写入，那其中的原始内容将被覆盖**，也就是说，你会丢失文件原有的信息；
- 就像完成文件读取后需要关闭 `Scanner` 一样，**当我们完成写入工作后，也别忘了关上 `PrintStream`**。

使用示范

让我们回到之前那个读取 .txt 文本文件的例子。

现在，我对程序做了一些改进，先让程序读取一遍文件内容，在我写入新文本后再重新读取一次，并查看修改文件的操作是否成功。

```
import java.io.File;
import java.util.Scanner;
import java.io.PrintStream;
public class fileReader {
    public static void main(String[] args) {
        try {
            // 设置文件路径
            File path = new File("C:/Users/Desktop/text.txt");
            // 第一部分：读取文件
            // 调用 Scanner 类
            Scanner fileScan = new Scanner(path);
            // 读取文件并显示内容
            String firstLine = fileScan.nextLine();
            System.out.println(firstLine);
            fileScan.close();
            // 第二部分：写入文件
            // 调用 PrintStream 类
            PrintStream writer = new PrintStream(path);
            // 声明待写入的文本
            String text = "我成功地写入内容啦！";
            // 将文本写入文件，关闭数据流
            writer.print(text);
            writer.close();
            // 第三部分：再次读取文件
            Scanner reader = new Scanner(path);
            firstLine = reader.nextLine();
            System.out.println(firstLine);
            reader.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

正如上文所说，如果对已经存在的文件进行写入操作，文件原有内容将被覆盖，并替换成新内容。

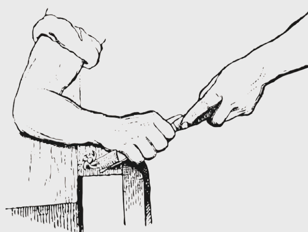
我们的电子存钱罐能够记住变化了



我们花了好几章时间来开发电子存钱罐，但其实它有一个设计缺陷：每一次存钱后，电子存钱罐会忘记之前的存款。在没有解决这个问题前，我们还不能放心地往里头存钱。

前面我们已经学会了读取和写入文件，现在是时候用上这些技能了。**我们要把表示存款的变量替换成文件，作为我们的数据库。**

硬编码！



硬编码（hard-code）是一种程序员在编程时偶尔会犯的坏毛病。一般来说，我们应该**把数据存储在外部文件中，在程序需要时再进行读取，而硬编码指的是直接把数据嵌入源代码中**。

这个习惯非常不好，因为一旦需要修改数据，我们就必须修改代码，有时会引发错误，也会让用户过于依赖编程者。此外，在我们翻译应用程序、编写程序的其他版本或是扩展它的功能时，也经常容易忘记修改嵌入代码的数据。

所以请牢记：今日偷懒**硬编码**，明日工作双倍还。

让我们以最新版本的电子存钱罐为基础，尝试如下修改。

(1) 把变量 `balance` 的初始值设为 `0`。

(2) 创建一个后缀名为 .txt 的文件，文本内容为一个数字（这是新的存款余额）。

(3) 编写一个读取文本内容的方法或一块代码，并将文本里的内容存储在 balance 变量中。这样一来，程序显示的余额就与文件的内容同步了。

(4) 编写一个方法或一块代码，将用户输入的存款金额加入余额变量，并将结果写入文本文件。这样一来，如果用户输入了存款金额，那当他离开程序时，文本文件的内容也会更新。

不要忘记，你需要使用 try/catch 语句才能通过编译。

为了减少你的工作量，我将最新版本的代码放在这儿，你可以在此基础上完成练习。基于同样的初始代码做练习，对答案的时候也更方便。

```
import java.util.Scanner;
public class MyPiggyBank {
    public static void main(String[] args) {
        String pass;
        int balance = 0;
        String requestPass = "请输入您的密码并按回车键";
        String passOK = "::::欢迎进入您的安心电子存钱罐。:::";
        String total = " >>> 你的余额为 ";
        String thx = "::: 感谢您使用电子存钱罐。 :::";
        String passKO = "密码错误。您无权使用该电子存钱罐。";
        // 创建新的 Scanner
        Scanner myScanner = new Scanner(System.in);
        System.out.println(requestPass); // 请求用户输入密码
        // 创建用来存储用户输入的变量
        pass = myScanner.next();
        // 检查密码是否正确
        switch (pass) {
            case "abracadabra":
                System.out.println(passOK + "\n" + total +
                    " " + balance);
                // 询问存款金额
                System.out.println("请输入存款金额");
                // 将存款金额存入一个新变量
                int deposit = myScanner.nextInt();
                if (deposit > 0) {
                    balance = balance + deposit;
```

```
                } else {
                    System.out.println("存款金额必须大于0。");
                    System.out.println("您的存款请求已被拒绝。");
                }
                // 显示最新余额
                System.out.println(total + " " + balance +
                    "\n" + thx);
                break;
            default:
                System.out.println(passKO);
                break;
        }
    }
}
```

解决方案

为了让你读起来更清楚，我把答案分成了几部分。

(1) 首先导入需要的类：File 和 PrintStream（Scanner 类已经完成了导入）。

```
import java.util.Scanner;
import java.io.*;
```

注意：.io 后面的星号（*）的作用是导入某个包中的所有类，这样就不用再分别导入 File 和 PrintStream 类了。

(2) 将 balance 变量的初始值设为 0。

(3) 声明 File 类的变量，然后声明文件路径：

```
File path = new File("C:/Users/Desktop/scripts/balancerecord.txt");
```

(4) 为了使用 PrintStream 类，加入 try/catch 语句：

```
try {
    switch (pass) {
        case "abracadabra":
            Scanner fileReader = new Scanner(path); // 新建 Scanner
            balance = fileReader.nextInt();
            System.out.println(passOK + "\n" + total + " " + balance);
            // 请求用户输入存款金额
            System.out.println("请输入存款金额");
            // 创建用来存储用户输入的变量
            int deposit = myScanner.nextInt();
            if (deposit > 0) {
                balance = balance + deposit;
            } else {
                System.out.println("存款金额必须大于 0。");
                System.out.println("您的存款请求已被拒绝。");
            }
            // 显示最新余额
            System.out.println(total + " " + balance + "\n" + thx);
            break;
        default:
            System.out.println(passKO);
            break;
    }
} catch (Exception e) {
}
```

注意：

- 我把 **switch** 语句块放到了 **try/catch** 代码块中；
- 我创建了一个新的 **Scanner** 对象（名为 fileReader），专门用来读取文件；
- **fileReader** 从文件中读取内容并赋值给 **balance** 变量；
- 我编写了 **catch** 语句块，但还没有在里面填入代码。

修改代码后，我再次确认了程序可以通过编译。现在，我完成了第一项重

要的修改：我的电子存钱罐不再只基于固定的余额运行，它终于能够从文件中读取数值了。下一步，我们要让电子存钱罐把余额写入文件。

现在，我着手第二部分：将余额写入文件。为此要进行以下操作。

(1) 在 if 语句块、紧接着更新 balance 变量值的地方，我用同一个文件路径作为参数，创建了 PrintStream 对象，并让它把 balance 变量的值写入该文件：

```
if (deposit > 0) {
    balance = balance+deposit;
    // 调用 PrintStream
    PrintStream writer = new PrintStream(path);
    writer.print(balance);
    writer.close();
}
```

(2) 有不少方法可以显示最新的余额。

a. 不做任何修改，保留目前的实现方式，因为显示的 balance 变量值和写入文件的数值一样。这也能从侧面验证你写入文件的数值是正确的。

b. 编写代码，比较文件内容是否和 balance 的值一致，并生成布尔值。如果布尔值为假（false），则发出警告。但这种方式有点复杂。

c. 修改 System.out.println 的参数，使该方法从文件里读取并显示余额值。这样也可以核对写入文件的内容。

我从以上实现方式中选择了第一种。如果你敢于探索，我建议你试着把以上三条全部用代码实现，作为实验，也作为挑战，来检验自己的编程水平。

现在，我来处理最后的一些细节问题。

(1) 我在 catch 语句块中加入了以下代码。当抛出异常时，程序可以对其进行处理并告知用户发生了什么：

```
} catch(Exception e) {
```

```
    System.out.println(e);
}
```

(2) 我需要确保数据流已被关闭。你可能已经注意到，在完成文件写入工作后，我们加上了 `writer.close()` 这条代码，但还没有关闭 `fileReader`。因此，在显示余额的代码后面，我加上了这一行：

```
fileReader.close();
```

最后的最后，再次检查程序是否能够通过编译。

现在，我们终于拥有了一个名副其实的、能够记住存款额的存钱罐啦！

终极练习：机密文件



让我们来挑战**最后一个练习**，它将结合我们在本书中学习过的几乎所有内容。这个终极练习也是全书**难度最高**的，你可以尝试从头到尾都自己完成，也可以使用我给你的代码片段。

在这个练习中，我们要创建一个**可以对消息进行加密和解密的程序**。这样，我们就可以保存密码、秘密消息，或是不想让“间谍”知道的其他内容。**如果朋友们也使用这个软件，大家就可以通过它交换加密信息。**

文本的加密是如何实现的

加密文本由一串字符组成，只有获取与之相对应的密钥才能解读，并明白每个字符实际对应的内容。如今存在许多不同的加密方法和加密标准，Java 也为加密操作提供了一些特定的库。但对现在的我们来说，这些库仍然有些复杂，所以不如借此机会回顾历史，了解一下凯撒加密法——它是世界上已知最古老的加密系统之一。

凯撒加密法



盖乌斯·尤利乌斯·恺撒是古罗马的第一位皇帝，正是

他发明了凯撒加密法（又称移位密码）。据说他当时就是运用这种加密方法向军队发送秘密消息的。

这个加密系统很简单：**将字母表上的所有字母都按照某个固定数目向后（或向前）移动，然后将原字母替换成位移后的新字母**。例如，在一个**偏移量为 2 的凯撒加密系统**中，原字母和对应的新字母如下所示。

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b

因此，如果我们用上面这个凯撒加密法加密字符串“program”，就会得到一个新的单词“rtqtco”。

我们要基于这个古老的加密系统**构建一个可以加密、解密消息的程序**。运行这个程序时，我们将使用一个文本文件，它在计算机文件系统中的位置是固定的。

编写完这个程序后，你可以把它分享给你的小伙伴。**你们可以用它相互发送加密消息，而且只有你们能使用程序把内容转换为可读文本。**

程序需求：

(1) 向用户**显示一个菜单**，让他选择“加密”或“解密”；
(2) **读取消息**，并按照用户的要求处理信息；
(3) 将处理后的内容**保存**在文件中；
(4) 在**屏幕上显示**处理后的内容；
(5) 我们的程序始终**与一个已有的特定文件相关联**。

因此，使用这个程序的步骤是：运行程序、输入信息、选择加密或解密选项进行处理。

程序算法

尝试一下自己编写这个程序的算法。你会在下一页看到我的参考答案。

参考算法

(1) 运行程序。

(2) 请求用户输入待处理的信息。

(3) 询问用户如何处理信息：1) 加密；2) 解密。

(4) 如果用户选择了 1)：

加密该信息；

将加密后的信息保存在预设的文件中。

(5) 如果用户选择了 2)：

解密该消息；

将解密后的消息保存在预设的文件中。

开始编程吧

为了让你更轻松地开展编程，你可以在下面看到我使用的类，以及我为开始编程做了哪些准备。

```
1  public class Encryptum {
2
3   // 这里我定义了一个字典，其中包含程序允许接受的所有字符
4
5  public static void main(String[] args){
6
7       // 一个代表偏移量的整数（int）变量
8
9       // 请求用户输入信息并将它存储在一个变量中
10
11      /* 询问用户对信息进行加密还是解密，
12      并将他的选择存储在一个整数变量中 */
13
14      /* 使用决策结构来调用相应的方法进行加密/解密，
15      并将结果写入文档中 */
16
17      // 加密方法
18
19      // 解密方法
20
21
22      }
23  }
```

我建议从一开始就保持代码的**整洁性**，创建特定的方法对信息进行加密和

解密，然后我们可以根据需要调用这些方法。

一些建议

- 创建一个**字符数组**作为字典，加密和解密方法都将使用这个字典。为了让你在任何地方都可以访问它，我们要在主方法**外**定义这个字符数组。如果还不会使用它，你也不用担心，之后我会告诉你具体的方法。
- **你需要使用位移量来定义一个整数变量**。可以将它设定为一个固定值，也可以每次都请求用户提供这个值。在我的程序中，我使用的位移量固定为 5。

以下就是我作为字典使用的字符数组：

```
static char[] chars = {
    'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
    'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
    'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
    'y', 'z', '0', '1', '2', '3', '4', '5',
    '6', '7', '8', '9', 'A', 'B', 'C', 'D',
    'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
    'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
    'U', 'V', 'W', 'X', 'Y', 'Z', '!', '@',
    '#', '$', '%', '^', '&', '(', ')', '+',
    '-', '*', '/', '[', ']', '{', '}', '=',
    '<', '>', '?', '_', '"', '.', ',', ' '
};
```

如何编写加密和解密的方法

虽然这些方法运行起来十分简单，但编写它们的确有难度。

- 将用户输入的**字符串转存到一个字符数组**中。
- **遍历字典**，**查找**数组中的每一个字符。
- 在字典中找到某个字符后，**将它替换为某位置相应的字符**。

我希望你可以尽量依靠自己的力量完成编程，因为无论结果如何，这都是一个有趣的练习。它迫使你绞尽脑汁设计方法，并且全力以赴测试程序。

如果你不想自己尝试，或者没能成功找到答案，这里有两种有效的**加密和解密方法**。不管怎么样，你还是需要让这些方法和自己的代码适配，确保一切都运行正常。

加密：

```
static String encrypt(String text, int shift) {
    char[] readableArray = text.toCharArray();
    for (int i = 0; i < readableArray.length; i++) {
        for (int j = 0; j < chars.length; j++) {
            if (j <= chars.length - shift) {
                if (readableArray[i] == chars[j]) {
                    readableArray[i] = chars[j + shift];
                    break;
                }
            } else if (readableArray[i] == chars[j]) {
                readableArray[i] = chars[j - (chars.length
                    - shift + 1)];
            }
        }
    }
    return String.valueOf(readableArray);
}
```

解密：

```
static String decrypt(String cipher, int shift) {
    char[] cipheredArray = cipher.toCharArray();
    for (int i = 0; i < cipheredArray.length; i++) {
        for (int j = 0; j < chars.length; j++) {
            if (j >= shift && cipheredArray[i] == chars[j]) {
                cipheredArray[i] = chars[j - shift];
                break;
            }
            if (cipheredArray[i] == chars[j] && j < shift) {
                cipheredArray[i] = chars[(chars.length -
                    shift + 1) + j];
                break;
            }
        }
    }
    return String.valueOf(cipheredArray);
   }
```

最终结果

这是我为练习编写的完整版代码，你的代码不需要和我的完全一样，重要的是能够实现相同的功能。

```
import java.io.File;
import java.util.Scanner;
import java.io.PrintStream;
public class Encryption {
    static char[] chars = {
        'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h',
        'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
        'q', 'r', 's', 't', 'u', 'v', 'w', 'x',
        'y', 'z', '0', '1', '2', '3', '4', '5',
        '6', '7', '8', '9', 'A', 'B', 'C', 'D',
        'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L',
        'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
        'U', 'V', 'W', 'X', 'Y', 'Z', '!', '@',
        '#', '$', '%', '^', '&', '(', ')', '+',
        '-', '*', '/', '[', ']', '{', '}', '=',
        '<', '>', '?', '_', '"', '.', ',', ' '
    };
    public static void main(String[] args) {
        // 设置偏移量
        int shift = 5;
        System.out.println("请输入信息并按回车键");
        Scanner reader = new Scanner(System.in);
        String text = reader.nextLine();
        System.out.println("进行哪项操作? ");
        int option = reader.nextInt();
        switch (option) {
            // 加密
            case 1: // 调用加密方法
                String enc = encrypt(text, shift);
                try {
                  File f = new File("C:/User/Desktop/
                      scripts/secret.txt");
```

```
                    PrintStream writer = new PrintStream(f);
                    writer.print(enc);
                    writer.close();
                } catch (Exception e) {}
                // 将结果写入文档，并在屏幕上显示
                System.out.println("加密文本: " + enc);
                break;
               // 解密
            case 2:
                String dec = decrypt(text, shift);
                System.out.println("解密文本: " + dec);
                break;
        }
    }
    static String encrypt(String text, int shift) {
        char[] readableArray = text.toCharArray();
        for (int i = 0; i < readableArray.length; i++) {
            for (int j = 0; j < chars.length; j++) {
                if (j <= chars.length - shift) {
                    if (readableArray[i] == chars[j]) {
                       readableArray[i] = chars[j + shift];
                        break;
                    }
                } else if (readableArray[i] == chars[j]) {
                    readableArray[i] = chars[j - (chars.
                        length - shift + 1)];
                }
            }
        }
        return String.valueOf(readableArray);
    }
    static String decrypt(String cipher, int shift) {
        char[] cipheredArray = cipher.toCharArray();
        for (int i = 0; i < cipheredArray.length; i++) {
            for (int j = 0; j < chars.length; j++) {
                if (j >= shift && cipheredArray[i] ==
                    chars[j]) {
```

```
                        cipheredArray[i] = chars[j - shift];
                        break;
                    }
                    if (cipheredArray[i] == chars[j] && j <
                        shift) {
                        cipheredArray[i] = chars[(chars.length
                            - shift + 1) + j];
                        break;
                    }
                }
            }
            return String.valueOf(cipheredArray);
        }
    }
```

进行测试

现在我们该对程序进行测试了。

尝试设计所有可能的使用场景，并验证在这些情况下会发生什么，比如说：

- 输入一段可读文本，并对它进行加密；
- 再输入一遍这段文本，对它进行解密；
- 输入一段带数字和大小写字母的文本。

优化程序

虽然练习没有涉及这部分内容，但其实**你可以对程序的一些细节进行优化。**可以选择现在就改进，也可以作为后续项目开展。

- **移动字符的偏移量可以是个变动值。**（刚刚我们将它设为一个**固定值**，好吧，这的确有点草率。）或者，在程序开始加密前，由用户来设定这

个偏移值。当你和好几个朋友一起使用这个程序时，只要每个人设定的密钥值不同，大家就只能阅读自己的信息了。

- 程序的进阶版可以**直接从文件而不只是键盘读取信息**。朋友们可以通过消息助手或邮件向你发送文件，程序运行后将直接定位这个文件并对它进行解密。你甚至可以添加一项功能，让程序显示文件列表供你选择，或者直接输入文件名进行搜索。
- **用户有时可能误操作**，比如选择了选项 3 或其他不存在的选项，我们还没有**对此添加任何保护措施**。

第6章

启程！学习的下一站



如果你跟随本书的脚步学到了这一章，那我要非常真诚地祝贺你！

因为你已经拥有了良好的知识基础，也在学习中证明了你的恒心和耐心。

编程能够发掘我们身上最优秀的品质，培养我们的毅力、逻辑思维和反思能力。在进展不顺的时候，也会让我们经受一些挫折，感到失落、无力，甚至失去信心……但别担心，**我们在学习的道路上总会遇到许多困难**，而且可能比我们想象得还要频繁。就算完成了学习，在今后的学习和应用中，也有很多挑战等着我们。哪怕是专业的程序员，每天也都会出错呢。

如果你想要继续深入学习 Java 编程，并且在思考接下来应该干些什么，我有两条建议要送给你：

(1) **再复习一遍**迄今为止所学的内容，养成最好的编程习惯；

(2) 如果已经顺利完成了第一点，你就可以继续学习**模块化**和**面向对象的编程**了。

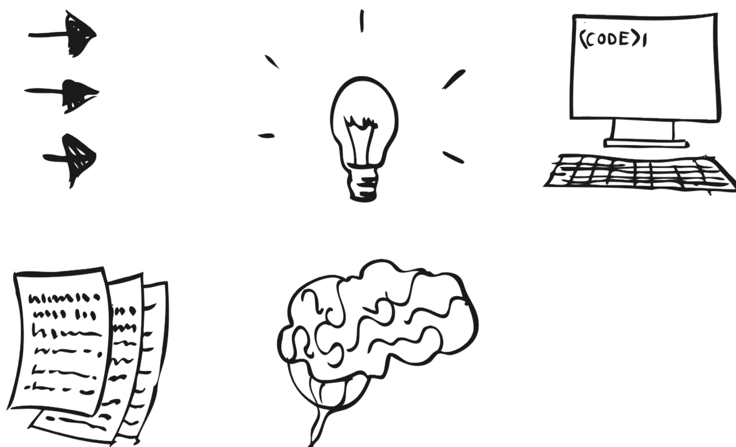
本书的目标是向初学者讲解基本的语法结构和数据类型，帮助他们培养一

种算法思维。如果你已经掌握了这些，本书就无法继续帮你学习了。

合上本书之前，我想再送给你一些有关**养成良好编程习惯**的建议，希望你能够每天实践它们。尽管这些建议现在听起来有点抽象，但我希望你能一边回顾迄今为止编写的所有程序，一边思考和领悟其中的道理。渐渐地你就会发现，还有许多可以改进的地方。写完程序后，你永远无法保证它就是最完美的版本。我建议你按照以下方法再检查一遍整个程序，设法让代码更清晰、更简洁、更高效。没有什么比写出**漂亮代码**更能让程序员满足的了。

避免重复代码

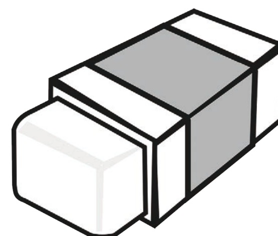
避免重复（DRY，don’t repeat yourself）是一项非常重要的原则，我相信你现在就可以在代码中按这条原则做了。**试着避免编写重复代码**。如果你发觉自己正在复制粘贴代码，也许就应该将这部分代码抽离出来了。把它放入一个方法，通过调用方法复用这段代码。遵循这个原则，我们的代码就会**更加稳定**且**易于维护**：在需要改动的时候，你**只需更改一个地方**，避免了在程序中一处一处地修改——这可方便多了。



其实你并不需要它

你知道吗，当我们收拾行李去度假时，我们带的东西是最后实际使用物品的两倍。编程的时候，我们也会犯这个错误。**其实你并不需要它**（YAGNI，you aren't gonna need it）这条原则告诉我们，**不要为了以防万一而添加不必要的功能**。一个程序的功能越多，它出错的概率就越高，也更难以维护。在设计程序的时候，**你只需要编写此刻应该实现的内容**。如果之后需要更多的功能，可以再升级程序。

童子军规则



童子军规则指的是，**离开营地后要比到达时更干净**。当你要改进一个现有程序，尤其是由别人创建的程序时，就可以应用这条规则。正如我们在前几章所学的，你需要**一遍遍地重构它**。

不要急着敲键盘

在开始编程前，要先记录、策划并设计相关内容。在有一个清晰的编程框架前，你可能需要无数遍地在脑中写下、擦掉想法。方法拆解和算法重构会帮助你节约不少时间。

永远要记得做测试

代码一旦通过了编译，大家就以为可以收工了，**但在完成必要的代码测试**

并得到不错的结果之前，一切还不能算大功告成。有时候就算编译正常，程序在运行时也会发生错误，或是出现其他意想不到的情况。你需要尝试列出所有的可能性，并一一测试。你不仅要测试需要实现的场景，也要测试那些可能失败的情况。要记住，用户们永远会用你从未想过的方法使用程序，他们是无法预测的。

致谢

我想感谢的人有许多。

首先，我要感谢阅读本书的你。

无论你是否喜欢这本书，如果你愿意在**亚马逊的产品页面**留下宝贵的评论，我将非常高兴[①]。你的留言将帮助我改进工作，并推动这个项目不断前进。

你可以在 GitHub 页面找到书中**所有练习的完整代码**。

感谢我亲爱的 Saïda 和 Leila。

感谢 Albert 对我的信任。

① 本书中文版的相关评论请提交至http://www.ituring.com.cn/book/2594。——编者注

这本轻松有趣的Java书
教你如何成为一名“魔法师”



学会踏入“魔法世界”的第一条咒语，让电脑屏幕显示你希望出现的文字

创造独一无二的电子存钱罐，存入自己的零花钱，还可以给它加上魔法保护罩，只有知道暗号的小伙伴才能打开



模拟火箭升空倒计时，电脑会听从你的命令从5数到1

通过魔法让计算机和你进行“石头剪刀布”对战，看看谁更胜一筹

加密你的机密文件，把它变成无人能破解的神秘文字，只有你知道真实含义

图灵社区：iTuring.cn
反馈/投稿/推荐邮箱：contact@turingbook.com
读者热线：(010) 51095186-600

分类建议	计算机/Java
	计算机/少儿编程

人民邮电出版社网址：www.ptpress.com.cn



ISBN 978-7-115-50763-1
定价：39.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱: ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育 : 好书、活动每日播报
微博 @图灵社区 : 电子书和好文章的消息
微博 @图灵新知 : 图灵教育的科普小组
微信 图灵访谈 : ituring_interview，讲述码农精彩人生
微信 图灵教育 : turingbooks